Visualiseren van VLAN-datastromen binnen het NDIX-netwerk

Afstudeerverslag
Versie 2.0

Afstudeerder	Alexander Johan (Sascha) Landegge
Bedrijf	NDIX BV
Bedrijfsbegeleider	Harry Loof
Opleiding	HBO-ICT Software Engineer
School	Saxion University of Applied Sciences
Stagebegeleider	Theo Lansink
Periode	September 2021 – April 2022

Colofon

Onderwijsinstelling

Onderwijsinstelling : Saxion University of Applied Sciences
: Afdeling ACT
: Opleiding HBO-ICT
: Richting Software Engineering
Adres : Van Galenstraat 19
Telefoon : 088 - 0196363

Afstudeerdocent : Theo Lansink
Telefoon : +31 (0) 6 51 29 82 80
E-mailadres : t.j.h.lansink@saxion.nl

Opdrachtgever

Afstudeerbedrijf : NDIX BV
: Zuiderval 64
: 7543 EZ Enschede

Telefoon : 053 711 41 50
Website : www.ndix.net

Afstudeerbegeleider : Harry Loof
Functie : Manager Automatisering
Telefoon : +31 (0) 6 11 92 23 31
E-mailadres : h.loof@ndix.net

Student

Naam : Alexander Johan (Sascha) Landegge
Telefoon : + 31 (0) 6 31 57 02 14
E-mailadres : saschalandegge@live.nl
Studentnummer : 445538

Inhoudsopgave

Tabellen

Figuren

Terminologie

In dit verslag wordt gebruik gemaakt van bepaalde termen waar niet iedereen bekend mee is. In deze tabel worden deze termijn kort toegelicht.

Term	Toelichting
LAN	Local Area Network
VLAN	Virtual LAN
NOC	Network Operating Center
sFlow	Sampled Flow
Interface	Een fysieke poort op een switch

Tabel 1: Terminologie

1. Inleiding

NDIX is bezig met het uitbrengen van een klantenportaal voor de gebruikers van NDIX. In dit klantenportaal kunnen klanten zelf taken regelen die voorheen door NDIX-medewerkers gedaan moesten worden. De klanten van NDIX kunnen VLAN's aanvragen. Dit zijn digitale LAN-connecties waarmee je locaties van bedrijven kan verbinden.

Voor klanten is het handig om in te kunnen zien wat hun dataverkeer is over hun afgenomen VLAN's. Naast dat dit voor klanten handig is, geeft dit essentiële inzichten bij het NOC waardoor ze het netwerk beter kunnen monitoren.

Tijdens mijn afstudeerperiode heb ik mij beziggehouden met het ontwikkelen van een dashboard voor zowel het NOC als de klanten zodat deze beter inzicht krijgen in het netwerk of hun afgenomen connecties. In dit document wordt gedocumenteerd hoe ik tot deze oplossing ben gekomen en hoe deze is gerealiseerd.

2. Organisatie

In dit hoofdstuk zal de benodigde achtergrondinformatie omschreven staan, hoe ziet de organisatie eruit, wat doet de organisatie en welke teams vallen daaronder?

2.1. Bedrijf

NDIX is een netwerkbedrijf dat in 2001 in Enschede is gestart. NDIX zorgt voor digitale connectiviteit tussen bedrijven in Nederland en Duitsland. Via glasvezel worden bedrijven aangesloten op het netwerk. In dit netwerk worden bedrijven onderling verbonden, zodat vestigingen gekoppeld worden en samen kunnen werken. Naast een netwerk levert NDIX ook een breed aanbod van ICT-diensten van meer dan 100 Nederlandse en Duitse leveranciers.

2.1.1. Diensten

NDIX levert via hun netwerk verschillende diensten. De diensten die ze aanbieden zijn afkomstig van externe partijen die ze leveren over hun netwerk. De vestigingslocatie van een bedrijf kan dan ook bepalen wat voor diensten er mogelijk zijn door het verschil in aanbod.

Bij de diensten kan je denken aan telefonie, internet, televisie, maar ook back-up, bewaking en sectorspecifieke diensten.

2.2. Organisatiestructuur

Binnen NDIX zijn er verschillende teams. Het team waar de afstudeerder de meeste interactie mee heeft is software. In dit hoofdstuk zijn twee relevantie teams benoemd. Informatie over andere teams binnen NDIX is te vinden in het plan van aanpak.

2.2.1. Software Engineering

Het softwareteam van NDIX is recentelijk opgericht. Het team is ontstaan uit het idee om de software intern te ontwikkelen. Voorheen was de software en de services die daarbij hoorden geoutsourcet of niet geautomatiseerd. Door de software intern te ontwikkelen kunnen er taken worden geautomatiseerd wat voorheen niet mogelijk was.

2.2.2. NOC

Het Network Operating Centre zorgt ervoor dat het NDIX-netwerk 24/7 stabiel blijft. Het team monitort dag en nacht of het netwerk en de verbindingen goed verlopen. Binnen het kantoor hebben ze een eigen muur met beeldschermen waar ze de datastromen kunnen inzien. Het NOC is het team wat gebruik gaat maken van het gerealiseerde product.

3. Opdrachtomschrijving

Het NOC-team bij NDIX controleert voortdurend het netwerk op abnormaliteiten en veranderingen. (Abrupte) verandering kan namelijk betekenen dat er een fout zit in het netwerk. De redenen hiervoor kunnen variëren van een kabelbreuk tot switch-instelling.

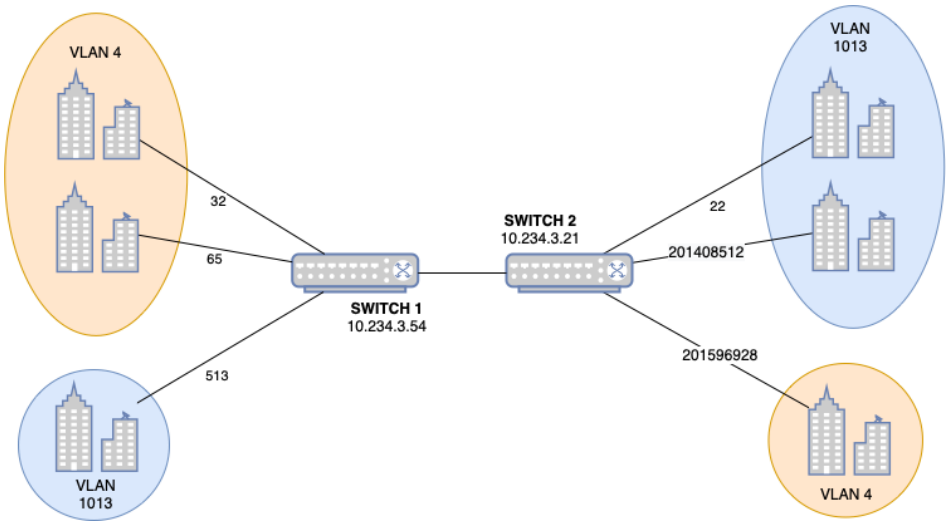
Het NOC heeft verschillende tools waar zij het netwerk mee controleren. Een tool die het NOC op dit moment heeft, is het dataverkeer in kaart brengen tussen de verschillende interfaces. Voor elke interface wordt er opgeslagen hoeveel data hier ontvangen/verzonden wordt. Deze gegevens worden voor een jaar opgeslagen zodat ook op lange termijn te zien valt welke trends er spelen op het netwerk.

3.1. Probleemstelling

Hoewel het NOC verschillende tools heeft om het netwerk te monitoren, hebben ze nog geen volledig overzicht van het netwerk. Op dit moment heeft het NOC een tool waarmee ze het dataverkeer kunnen inzien van een interface, bijvoorbeeld interface-index **201408512** op switch **10.234.3.21** (zie figuur 1).

Omdat het NOC ervoor is om het netwerk te monitoren en analyseren is het van belang dat zij een volledig overzicht hebben van het dataverkeer, dus ook VLAN en switch dataverkeer.

Een tool die het NOC nog niet heeft is het in kaart brengen van het VLAN en switch dataverkeer. Op het NDIX-netwerk zijn veel VLAN's actief en daarom is het voor het NOC belangrijk dat deze in kaart worden gebracht.



Figuur 1: VLAN-topografie

Met het in kaart brengen van de ±9500 VLAN's kan het volgende worden gerealiseerd:

- **Betere kennis van het netwerk**
  Omdat klanten gebruik maken van een VLAN en niet van een interface kan dit verduidelijking geven in het netwerk. Het bekijken van VLAN-data is in die zin een laag dieper.
- **Sneller problemen op het netwerk analyseren**
  Wanneer je een beter overzicht hebt van het netwerk kun je sneller bij het probleem komen.
- **Sneller problemen op het netwerk verhelpen**
  Een snellere analyse betekend dat problemen sneller verholpen kunnen worden.
- **Trends monitoren en daarop inspelen**
  Een stijgende of dalende trend van een VLAN kan de klant inzicht geven in veranderingen op het netwerk.

3.2. Opdracht

De opdrachtgever, NDIX, wil een tool waarmee zij het VLAN-dataverkeer in kaart kunnen brengen. Het in kaart brengen van het dataverkeer wordt gedaan door het te visualiseren in een dashboard welke geïmplementeerd moet worden binnen het NDIX-klantenportaal.

De huidige oplossing die wordt gebruikt om het dataverkeer van de interfaces te visualiseren kan niet worden gebruikt door het grote aantal VLAN's. De huidige oplossing werkt door middel van RRD-bestanden die voor elke interface worden aangemaakt. De ruimte die nodig is voor het opslaan van al deze bestanden is 75GB. Door de grote hoeveelheid VLAN's die het NDIX-netwerk bezit is het daarom niet mogelijk deze oplossing te gebruiken omdat dit zal resulteren in een opslagprobleem. De interface oplossing heeft er echter wel voor gezorgd dat er een protocol is geïnstalleerd op alle switches om data te verzamelen.

Het protocol dat op alle switches staat geïnstalleerd, is het sFlow protocol. sFlow staat voor Sampled Flow. Dit protocol zorgt ervoor dat er om de 10.000, 20.000 of 40.000 pakketjes een pakketje wordt onderschept en verzonden naar de sFlow collector. Deze data kan vervolgens binnen de collector worden opgeslagen en/of verwerkt. Het opslaan van deze data maakt het mogelijk om het inzichtelijk te maken in een dashboard.

Op dit moment wordt dit protocol gebruikt om de interfaces in kaart te brengen. Echter kan dit protocol ook gebruikt worden om VLAN's in kaart te brengen. De opdracht implementeert daarom ook een eigen 'collector' waar de VLAN-data uit afgelezen kan worden.

De wens is om deze data vervolgens op te slaan en te visualiseren in een dashboard. Het dashboard dient vervolgens te worden geïmplementeerd in het bestaande klantenportaal van NDIX. Door het dashboard ook beschikbaar te stellen in het klantenportaal kan niet alleen het NOC maar ook klanten bij de visualisaties van het netwerk.

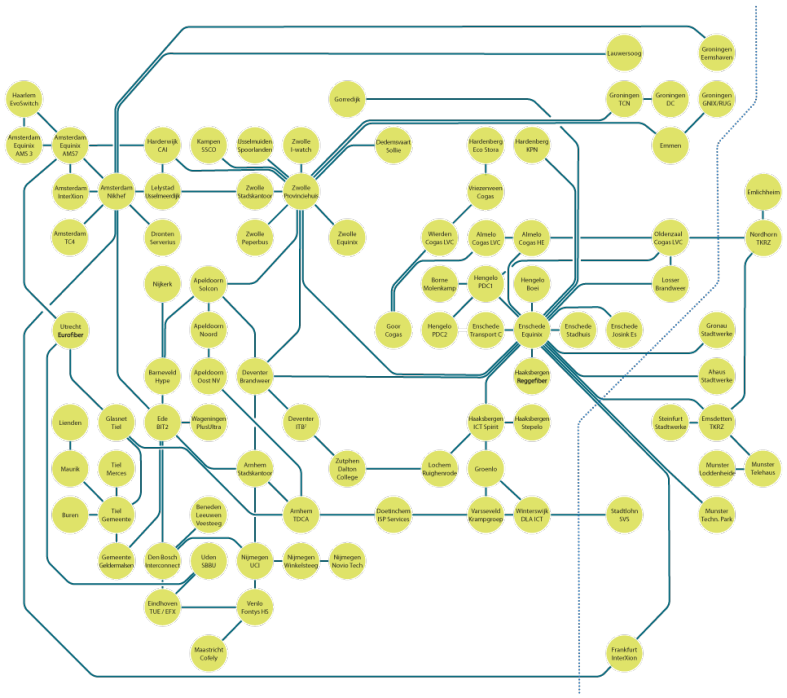
4. Vooronderzoek

In dit hoofdstuk wordt er antwoord gegeven op de volgende onderzoeksvragen:

- Hoe ziet het NDIX-netwerk eruit?
- Wat doet het sFlow protocol precies?
- Hoe wordt op dit moment de sFlow data gebruikt?
- Requirementsanalyse
- Hoe kan deze sFlow data het beste worden verzameld?
- Hoe kan deze sFlow data het beste worden opgeslagen?
- Wat is een Time Series Database?
- Welke Time Series Database is het meest geschikt?
- Hoe moet de sFlow data worden gevisualiseerd?

4.1. Hoe ziet het NDIX-netwerk eruit?

Het netwerk van NDIX bestaat uit verschillende onderdelen. Zo verbindt het netwerk verschillende locaties door Nederland en Duitsland (zie figuur 2). Deze locaties zijn onderling verbonden zodat ze vanuit deze locaties verschillende bedrijventerreinen kunnen aansluiten op hun netwerk.



Figuur 2: NDIX netwerk

Switches en Interfaces

NDIX heeft op hun netwerk een groot aantal switches. Deze switches worden gebruikt om fysiek apparaten te verbinden met het NDIX-netwerk. Een switch fungeert als verdeelkast voor de inkomende verbindingen vanuit het NDIX-netwerk. De switch zorgt ervoor dat de data naar de juiste poorten/apparaten wordt gestuurd. De poorten die op een switch zitten worden interfaces genoemd.

VLAN

NDIX biedt de mogelijkheid om op hun netwerk VLAN's aan te vragen. Dit is een virtuele variant van een LAN (Local Area Network). Een VLAN maakt het mogelijk om verschillende locaties veilig met elkaar te verbinden. Bij NDIX verbinden ze klantlocaties door middel van deze VLAN's. Door deze klantlocaties te verbinden met een VLAN krijgen deze een onderlinge verbinding die niet over het internet fungeert maar over de datalink laag (laag 2) van het netwerk.

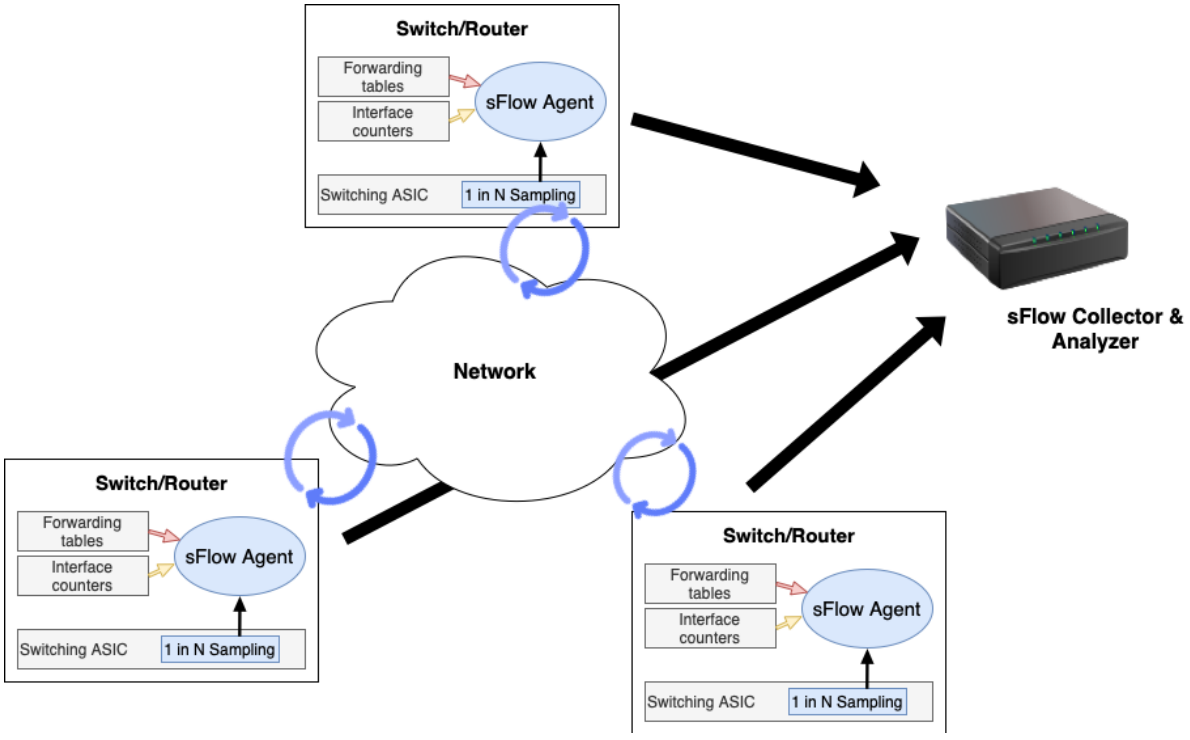
Doordat de VLAN niet via het internet fungeert is deze voor de buitenwereld niet toegankelijk. Dit maakt dat het gebruik van een VLAN aanzienlijk veiliger is dan bijvoorbeeld het gebruik van een VPN. Een VPN fungeert namelijk wel via het internet en is daardoor kwetsbaarder voor aanvallen van buitenaf.

Het NDIX-netwerk beschikt over duizenden interfaces en per interface kunnen er meerdere VLAN's aanwezig zijn.

4.2. Wat doet het sFlow protocol precies?

Het protocol dat wordt gebruikt om de data binnen te krijgen van de switches op het netwerk heet Sampled Flow, oftewel sFlow. Dit is een industrie standaard die wordt gebruikt voor het monitoren van (grootschalige) netwerken, foutoplossing en gebruik van dataverkeer.

sFlow bestaat uit twee onderdelen, een sFlow Agent en een sFlow Collector. Deze onderdelen werken samen om een duidelijk beeld te geven van de data die over het netwerk heen stroomt.



Figuur 3: sFlow diagram

4.2.1. Agent

De sFlow agent verstuurt informatie naar een sFlow collector. In dit geval is de sFlow-agent de switch op het NDIX-netwerk. Deze switch zorgt ervoor dat er om de aantal seconden een update wordt verstuurd met informatie over het dataverkeer van deze switch.

De informatie die de sFlow agent verstuurd wordt door middel van *packet sampling* gecreëerd. Deze Packet Sampling zorgt ervoor dat er in een klein bericht kan worden omschreven wat er over de agent, aan data, verstuurd is.

4.2.1.1. Packet Sampling

Packet Sampling zorgt ervoor dat door middel van steekproeven het mogelijk is om met een fractie van de data alsnog een accuraat beeld te krijgen van de gegevens die over het netwerk gaan. Dit verkleinde pakket is nooit 100% accuraat, maar is wel voldoende accuraat om gebruikt te worden als meet-tool (Phaal & Panchen, sd). Deze methode is goed toepasbaar voor grote datastromen omdat dit maar een fractie van de daadwerkelijke data doorstuurt.

Voorbeeld
Stel je voor dat er 1.000.000 pakketjes over het netwerk heen gestuurd worden. Dan verzamel je door middel van packet sampling van deze pakketjes 2% (20.000 pakketjes) en stuur je deze door. In dit verstuurde bericht is dan te zien wat de verhoudingen zijn binnen het pakket. Deze verhoudingen zijn statistisch accuraat genoeg om gebruikt te worden in gemaakte oplossingen.

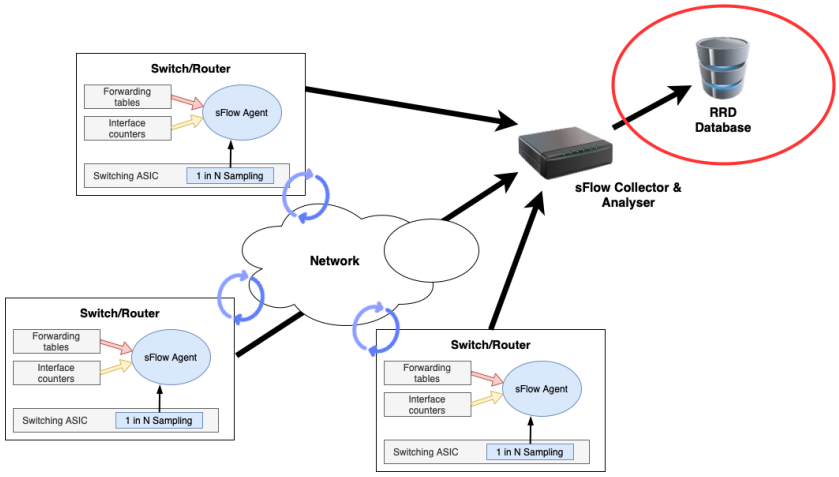
Tijdens het vooronderzoek is geconstateerd dat de packet sampling op de gegeven test server bij NDIX instellingen van 10.000, 20.000 en 40.000 bevat. Deze waardes verschillen van elkaar omdat er verschillende netwerksnelheden op het NDIX-netwerk zijn. Deze verschillende netwerksnelheden zorgen ervoor dat er meer data overheen gestuurd kan worden en zo meer data verzameld kan worden. Door de packet sampling instelling te verhogen verwerkt de server minder pakketjes omdat deze eens in de 10/20/40 duizend pakketjes gegevens verzameld.

4.2.2. Collector

Een sFlow collector is een server waar de sFlow agent data pakketjes heen stuurt. De sFlow pakketjes komen bij de collector binnen op poort 6434. Op deze server kunnen deze pakketjes vervolgens worden onderschept en gebruikt om data inzichtelijk te maken of op te slaan.

4.3. Hoe wordt op dit moment de sFlow data gebruikt?

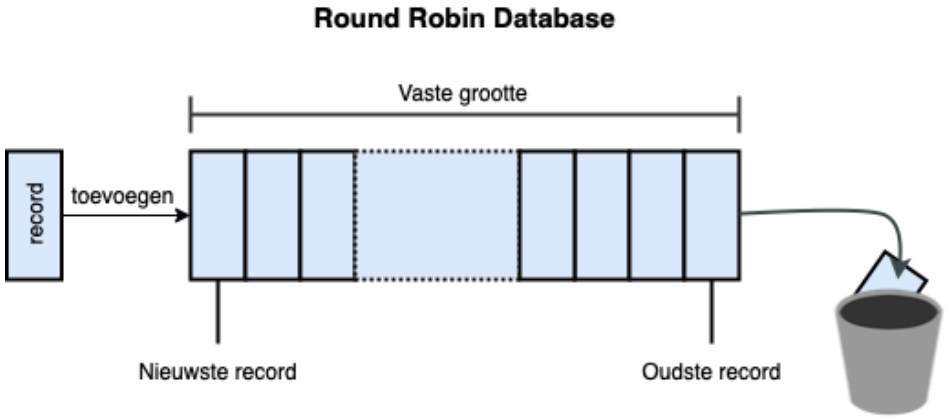
Op dit moment wordt er een sFlow collector gebruikt om het dataverkeer van de interfaces in kaart te brengen. Deze collector is door NDIX zelf ontwikkeld en schrijft de data naar een RRD (Round Robin Database) bestand (zie figuur 4).



Figuur 4: Huidige situatie – RRD sFlow collector

4.3.1. RRD

Een RRD-bestand is een database met een vaste grootte. Deze grootte wordt bij het aanmaken vastgesteld door in te stellen hoeveel datapunten er opgeslagen mogen worden. En over een lange periode ook wat voor gemiddeldes deze moet genereren.
Wanneer er nieuwe data naar deze database wordt geschreven, wordt de laatste record van de database verwijderd. Vandaar ook de naam “round’ robin database. De data circuleert zodoende door het bestand. In figuur 5 is te zien hoe de data wordt toegevoegd en verwijderd.



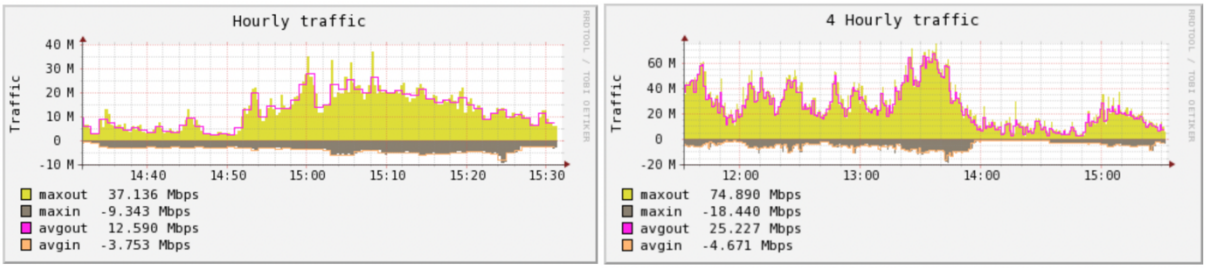
Figuur 5: Werking Round Robin Database

Waardes
De sFlow collector ontvangt sFlow pakketjes en bekijkt de meegestuurde waardes. De waardes *ifOutOctets*, *ifOutDiscards* en *ifOutBroadcastPkts* worden opgeslagen en wegeschreven naar een gegenereerd RRD-bestand.

In het *ifOutOctets* RRD-bestand zijn waardes te vinden van BytesIn en BytesOut. Dit zijn de hoeveelheid bytes die de interface heeft ontvangen/verstuurd.

4.3.2. Visualisaties

Deze waardes zijn inzichtelijk gemaakt op een interne website van NDIX waar switches en bijbehorende interfaces te zien zijn. De gebruiker kan, wanneer deze een switch geselecteerd heeft, een keuze maken uit te de bijbehorende interface en hiervan zijn data inzien. In figuur 6 is te zien hoe deze data weergegeven wordt. Op de interne website zijn grafieken te zien voor dataverkeer per *minuut*, *uur*, *4 uur*, *dag*, *week*, *maand* en *jaar*.



Figuur 6: RRD data-visualisatie (Hourly, 4 Hourly)

4.4. Requirementsanalyse

De afstudeerder heeft functionele en niet functionele requirements opgesteld in samenwerking met de bedrijfsbegeleider met oog op een realistisch uitvoerbare opdracht waar het bedrijf ook daadwerkelijk baat bij heeft.

De requirements zijn opgesteld door middel van een interview die gehouden is met de bedrijfsbegeleider. In dit interview is duidelijk geworden wat de wensen zijn van het bedrijf en hoe deze het beste tot stand konden komen.

Bij het opstellen van de requirements is er rekening gehouden met de *MoSCoW* prioritering (van Vliet, 2008). Deze prioritering verdeeld de requirements in: must, should, could en won't.

4.4.1. Must

Deze requirements moet het eindproduct te bevatten.

ID	Toelichting	F/NF
M01	Het systeem kan in 'real time' data uitlezen.	Functioneel
M02	Het systeem kan per VLAN de data opvragen.	Functioneel
M03	Het systeem kan per switch de data opvragen.	Functioneel
M04	Het systeem kan per interface de data opvragen	Functioneel
M05	De gebruiker moet zijn eigen VLAN-data kunnen inzien.	Functioneel
M06	Het NOC moet de data per switch kunnen inzien.	Functioneel
M07	Het NOC moet de data per interface kunnen inzien.	Functioneel
M08	Het NOC moet de data per VLAN kunnen inzien.	Functioneel

M09	De visualisaties dienen worden weergegeven in het klantenportaal.	Functioneel
M10	De frontend dient geschreven te worden in VueJS.	Niet Functioneel
M11	De backend dient geschreven te worden in het PHP framework Symfony.	Niet Functioneel
M12	Het systeem moet voldoen aan de bedrijfsstandaard voor beveiliging.	Niet Functioneel
M13	De frontend dient geïmplementeerd te worden met de huisstijl van het afstudeerbedrijf.	Niet Functioneel

Tabel 2: Must requirements

4.4.2. Should

Deze requirements zou het eindproduct moeten bevatten. Het is zeer wenselijk, maar niet noodzakelijk.

ID	Toelichting	F/NF
S01	De geschreven backend code is getest met behulp van PHPUnit tests.	Functioneel
S02	Het systeem moet worden gebouwd als een service binnen het NDIX-api project.	Niet functioneel
S03	De sFlow-data dient voor een jaar bewaard te blijven.	Functioneel

Tabel 3: Should requirements

4.4.3. Won't

Deze requirements zullen niet in deze opdracht uitgevoerd worden. Wel is het handig om deze requirements mee te nemen bij een vervolgproject.

ID	Toelichting	F/NF
W01	De gebruiker kan meldingen instellen per VLAN	Functioneel
W02	De gebruiker kan meldingen instellen per interface.	Functioneel
W03	De gebruiker kan meldingen instellen per switch	Functioneel

Tabel 4: Won't requirements

4.5. Hoe kan deze sFlow data worden verzameld?

Om de sFlow data op te kunnen slaan moet het eerst worden ontvangen op een sFlow collector. Er zijn verschillende kant-en-klare sFlow collectoren voor verschillende use-cases. Voor de opdracht is het van belang dat de sFlow data in 'real time' wordt weergegeven.

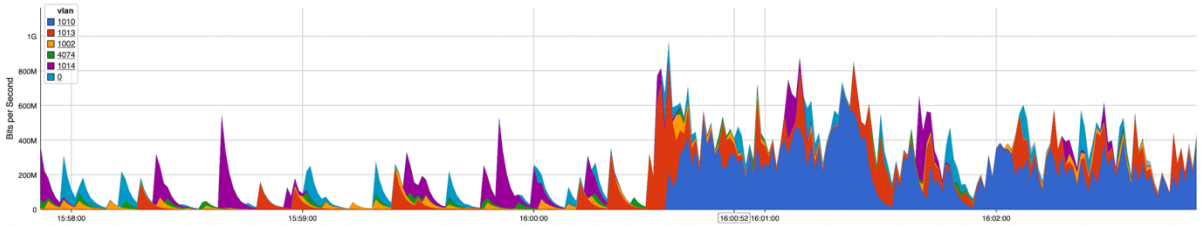
Tijdens verder onderzoek naar het sFlow protocol biedt de website van het protocol meerdere voorbeelden van implementaties (InMon Corp., sd). Daarnaast staat er op de website ook een lijst

met sFlow collectoren. In een van deze voorbeelden van implementaties wordt gebruik gemaakt van de sFlow-RT collector. Deze collector biedt real time inzicht in het sFlow dataverkeer.

4.5.1. sFlow-RT

sFlow-RT is een open source sFlow collector van het bedrijf InMon Corp. Dit bedrijf is tevens ook de ontwikkelaar van het sFlow protocol. Deze collector verzorgt een platform waarop andere sFlow-RT applicaties op kunnen draaien. De RT in de naam staat ook voor “real time” wat nodig is voor de opdracht.

Het installeren van sFlow-RT is relatief makkelijk door het gebruik van Docker containers. sFlow-RT heeft, naast zichzelf, ook veel applicaties in docker containers beschikbaar. Door het installeren van sFlow-RT en de bijhorende ‘browse-flows’ applicatie is het mogelijk om in ‘real time’ data in te zien. sFlow is een uitgebreid protocol met veel verschillende metrics die uitgelezen kunnen worden. De applicatie ‘browse-flows’ heeft de mogelijkheid om te filteren op VLAN-data wat in figuur 7 te zien is.



Figuur 7: 'real time' vlan-data

In figuur 7 is te zien dat er op de collector vanuit verschillende VLAN’s data wordt ontvangen. Deze oplossing biedt ‘real time’ inzicht in de VLAN-data. Echter slaat deze applicatie geen data op voor een langere tijd. De grafieken die worden weergegeven zijn dan ook alleen data wat in de huidige sessie ontvangen is.

De applicatie sFlow-RT heeft ook een API ter beschikking waar de ontvangen data op te vragen is. Deze API heeft verschillende end-points voor verschillende datapunten. De beschikbare API-calls zijn bijvoorbeeld: agents, metrics, flows en utilities. Met het metrics end-point is het mogelijk om geformatteerde data terug te krijgen en vervolgens op te slaan in een database. Het opvragen van data door middel van een API-call heet *pull*.

sFlow-RT biedt meerdere opties aan om de geformatteerde data op te slaan of te verwerken in andere programma’s. In hoofdstuk 4.8 staat beschreven naar welke databases onderzoek is gedaan voor de mogelijke opslag. Een van deze databases heeft een geïmplementeerde sFlow collector genaamd Telegraf.

4.5.2. (InfluxDB) Telegraf

Op zoek naar een sFlow collector kwam, tijdens het onderzoek naar de mogelijkheden van sFlow-RT, de database InlfuxDB naar voren als mogelijke optie voor opslag. Het unieke aan deze database is dat deze ook een eigen server agent heeft die statistieken kan verzamelen van verschillende apparaten. Deze server agent heet Telegraf. Voorbeelden van apparaten waar Telegraf statistieken van kan verzamelen zijn: IOT-oplossingen, server gegevens en sFlow data.

Het laatste punt heeft betrekking op de data die in deze opdracht van belang is. Een van de beschikbare plugins van Telegraf heeft namelijk dezelfde functionaliteit als een sFlow collector. De

manier van data wegschrijven is echter wel fundamenteel anders dan bij een *scraper,* zoals bij sFlow-RT. Telegraf schrijft namelijk met de sFlow plugin data direct naar de database. Deze methode van data wegschrijven heeft *push*.

Voordat Telegraf data wegschrijft is het ook mogelijk om deze data te verkleinen, filteren en/of aggregeren. Het filteren en aggregeren van deze data is noodzakelijk omdat de hoeveelheid data die binnenkomt vanuit de sFlow collector te veel is om allemaal op te slaan in de database.

In figuur 8 is te zien hoe Telegraf als collector fungeert.

Figuur 8: sFlow visualisatie flow i.c.m. telegraf

In hoofdstuk 4.8 wordt gekeken naar welke databases gebruikt kunnen worden voor de opslag van sFlow-data. Er wordt daarom nu nog geen keuze gemaakt uit de verschillende sFlow collectoren.

4.6. Hoe kan deze sFlow data het beste worden opgeslagen?

Om de sFlow-data ook historisch in te kunnen zien dient deze opgeslagen te worden. De huidige oplossing voor het opslaan van sFlow data is het gebruik van RRD-bestanden. Deze oplossing is voor de interfaces te gebruiken omdat er hier +/- 6500 aantal van zijn. Per interface wordt er een apart RRD-bestand aangemaakt met een vaste grootte. De hoeveelheid VLAN's is aanzienlijk hoger. Om die reden kan er een opslagprobleem voorkomen wanneer er per VLAN een RRD-bestand wordt aangemaakt.

De beperkingen die een Round Robin Database heeft zijn:

- Met de huidige methode niet mogelijk om VLAN-verkeer in te zien.
- Voor elke interface een apart bestand, het openen/opslaan/sluiten van deze bestanden kost ook tijd en is afhankelijk van de schijf lees/schrijf snelheid.
- Voor de hoeveelheid VLAN's niet te gebruiken wegens opslag grootte.

Door deze beperkingen moet er worden gezocht naar een alternatief voor RRD-bestanden.

Een alternatief voor RRD-bestanden is een Time Series Database (Deri, Mainardi, & Fusco, 2012). Dit type database is geoptimaliseerd voor data dat vooral op tijd gebaseerd is. Voor constante datastromen zoals ontvangen wordt bij de sFlow collector is dit type database het beste alternatief op RRD-bestanden.

4.7. Wat is een Time Series Database (TSDB)?

Een Time Series Database is een database die zijn waardes opslaat door paren te maken van tijd en waardes. Een tsdb is anders dan een relationele database. De database bevat namelijk geen relaties. Ook is het normaal dat in een tsdb data niet oneindig wordt bewaard. Door de hoeveelheid en de aard van de data is het gebruikelijk dat deze na een periode wordt verwijderd.

Doordat de data uniform is, het is immers een tijd en waarde paar, is het mogelijk om geoptimaliseerde algoritmes te gebruiken om de data te comprimeren. Time Series Databases zijn binnen de IT een nieuwe opkomende trend omdat er steeds vaker toepassingen voorkomen waar grote stromen aan tijd gerelateerde data opgeslagen moet worden (Wayner, 2021).

Er zijn verschillende tsdb's beschikbaar die verschillen in functionaliteit en daarom voor verschillende toepassingen te gebruiken zijn.

4.8. Welke Time Series Database is het meest geschikt?

De verschillen tussen de Time Series Databases zorgt ervoor dat verschillende databases voor verschillende toepassingen gebruikt kunnen worden. Tijdens het onderzoek zijn er verschillende databases gevonden.

InfluxDB
InfluxDB is een database die inmiddels door veel bedrijven gebruikt wordt. Het begon als een open source project die is uitgegroeid naar een open source versie en een cloud versie die voor de klant gehost wordt. De database maakt gebruikt van z'n eigen Flux taal (influxdata, 2022) om query's mee uit te voeren en heeft verschillende functionaliteiten beschikbaar om data te aggregeren en te verwijderen op lange termijn.

Timescale DB
Deze database is volledig geïntegreerd met PostgreSQL. Dit zorgt ervoor dat deze database naast time-series data ook relationele data kan opslaan en relaties kan aanhouden binnen de database. Het gebruiken van Timescale DB in combinatie met PostgreSQL zorgt ervoor dat time-series data tot wel 20 keer sneller kan worden geschreven.

Prometheus
Prometheus slaat alle data op met een timestamp als key. De database heeft een reeks aan standaard query's die gebruikt kunnen worden om veranderingen in dataverkeer te analyseren. De database gebruikt PromQL om query's uit te voeren welke erg lijkt op GraphQL. Dit maakt het voor ontwikkelaars makkelijk om deze taal te gebruiken.

Van deze opties zijn InfluxDB en Prometheus de meest relevante opties om te vergelijken. De sFlow data hoeft namelijk niet opgeslagen te worden in een relationele database en deze twee opties bieden voor time-series data de meest efficiënte variant van dataopslag.

4.8.1. InfluxDB vs Prometheus

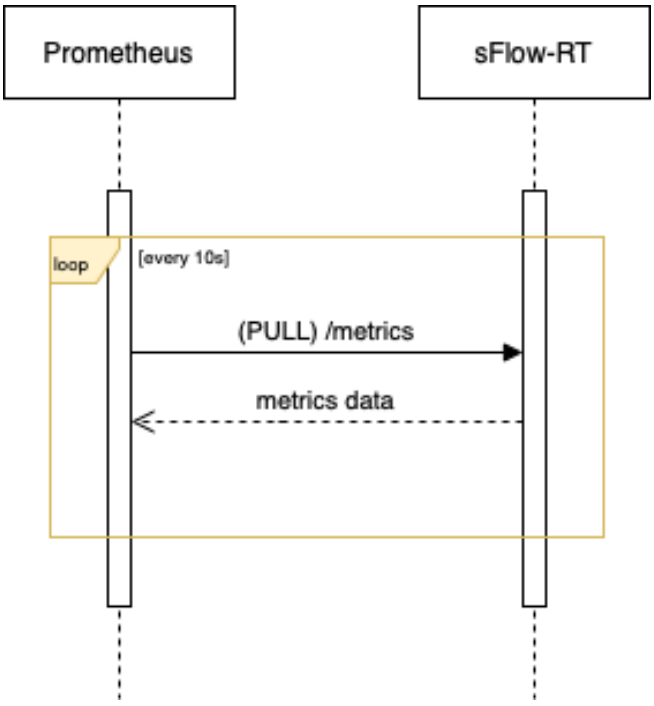
InfluxDB en Prometheus zijn beide time series databases. Beide databases hebben onderlinge verschillen die ervoor zorgen dat de een meer geschikt is voor deze casus dan de ander.

4.8.1.1. Prometheus

Prometheus is een open source time series database die veel wordt gebruikt in het werkveld. De database wordt gebruikt voor het monitoren en waarschuwen van gebeurtenissen. Een gebeurtenis is bijvoorbeeld wanneer een datastroom stopt. Dit zou dan kunnen wijzen op een storing op het netwerk. De database heeft de mogelijkheid om in 'real time' statistieken vanuit verschillende bronnen te verzamelen. De methode waarop Prometheus zijn data verzameld is de *HTTP pull/poll* methode.

Dataverzameling
De *pull* methode van data verzamelen heeft een aantal voor- en nadelen. Een van de voordelen is dat je op een vaste interval data kan verzamelen. Het is bijvoorbeeld mogelijk om elke 10 seconden een update te vragen aan de databron om vervolgens deze data op te slaan (zie figuur 9).
Door op deze manier data op te vragen met een vaste interval weet je altijd wat voor data je kan verwachten en op welke momenten dit aangeleverd wordt.

Het is echter ook mogelijk dat de server een te grote belasting krijgt door de grote hoeveelheid aanvragen die deze krijgt voor het opvragen van de data. Ook is het mogelijk dat de aanvraag een te grote hoeveelheid data moet terugsturen. Afhankelijk van de databron en instellingen wordt er bij het opvragen van data met een vaste interval gekeken naar de afgelopen datapunten of naar de huidige waarde.

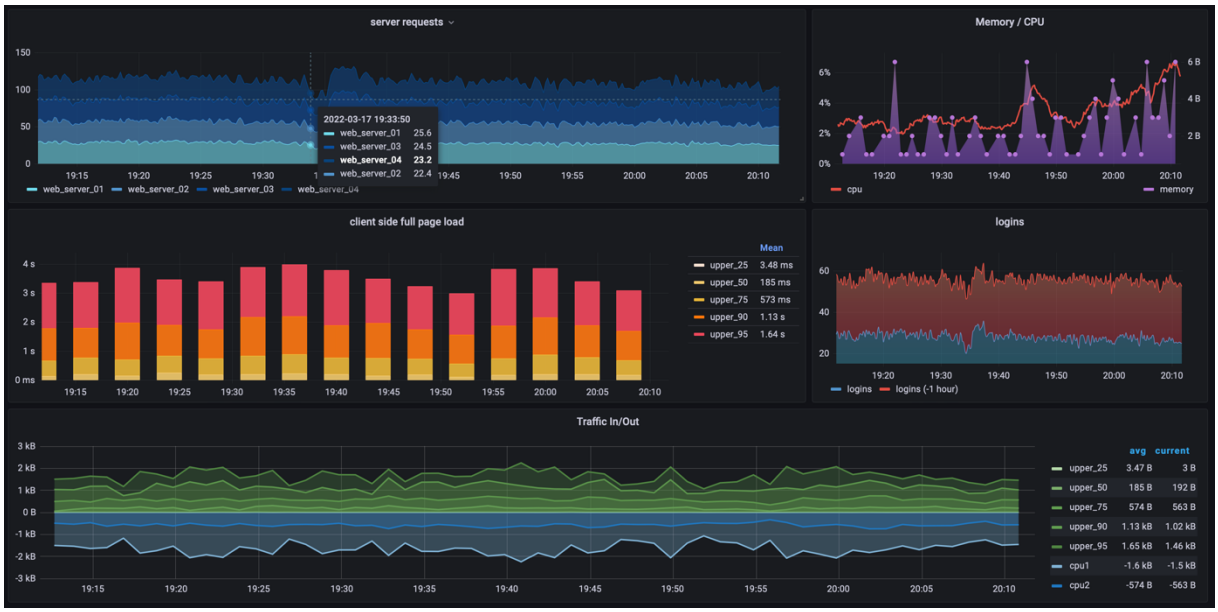


Figuur 9: Prometheus Pull

Een alternatief van de *pull* methode is de *push* methode. Bij deze methode wordt er niet met een vaste interval data verzameld, maar wordt er constant data naar de database geschreven. Dit heeft als voordeel dat je alle data binnenkrijgt, maar dit kan er dan ook voor zorgen dat de hoeveelheid hiervan te groot is.

Visualisaties
Prometheus integreert goed samen met Grafana. Grafana is een visualisatie tool waarmee het mogelijk is om dashboards te maken vanuit een Prometheus database. De tool zelf heeft geen data opgeslagen, deze vraagt hij op vanuit een Prometheus database.

In figuur 10 is een voorbeeld te zien van een dashboard die in Grafana is gemaakt.



Figuur 10: Voorbeeld Grafana dashboard

4.8.1.2. InfluxDB

Een alternatief van Prometheus is InfluxDB. Deze database lijkt op veel vlakken op Prometheus, echter zijn er wat verschillen qua mogelijkheden en visualisaties.

Dataverzameling
Het verschil wat InfluxDB met Prometheus heeft, is dat InfluxDB ook kan werken met de *push* methode. In tegenstelling tot de *pull/poll* methode schrijft deze methode constant data naar de database. Dit heeft als voordeel dat je alle data op de database binnenkrijgt. Dit kan van pas komen wanneer de gebruiker de data in veel details wil inzien/analyseren. Ook is het hierdoor mogelijk om de kleinste fluctuaties in de datastromen in te zien.

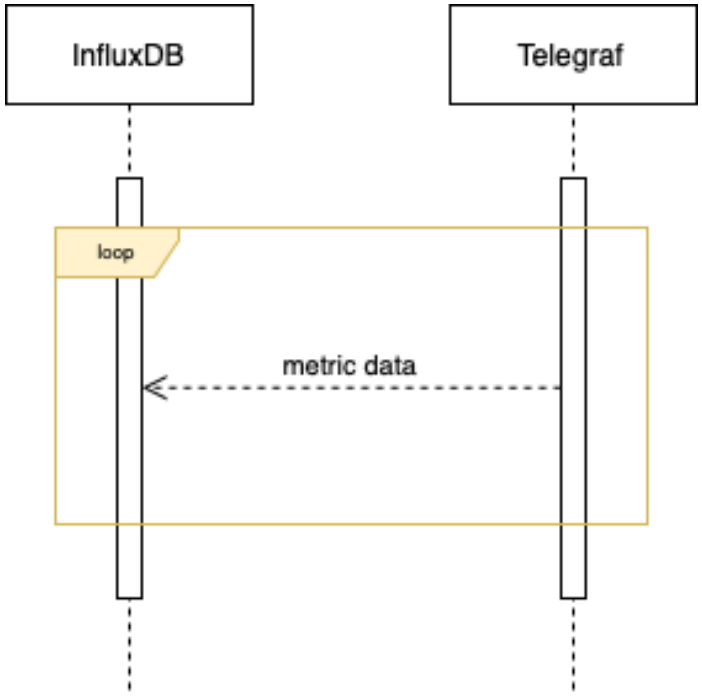
InfluxDB heeft ook de mogelijkheid om data op te vragen met de *pull* methode door middel van een ‘scraper’. Deze scraper zorgt ervoor dat er om de 10 seconden een *pull* request wordt gedaan naar de opgegeven URL waar deze de data kan opvragen.

Naast dat het een voordeel is dat InfluxDB door middel van de *push* methode alle data binnenkrijgt, kan het daarnaast ook een groot nadeel zijn. Doordat de database alle data ontvangt kan het zijn dat er te veel data binnen komt er daarom de database snel vult in opslag.

Zoals in hoofdstuk 4.5 besproken, bestaat er een sFlow collector genaamd Telegraf. Deze collector werkt samen met InfluxDB. Zo kan Telegraf ervoor zorgen dat niet alle data naar de InfluxDB database wordt gestuurd en kan hierdoor de datastroom verminderen. Denk hierbij aan dat Telegraf

bijvoorbeeld alleen de relevante data doorstuurt naar InfluxDB en zodoende aanzienlijk minder data hoeft te verwerken.

In figuur 11 is te zien hoe Telegraf zijn data naar InfluxDB pusht. Dit is een constante stroom van data naar de database.

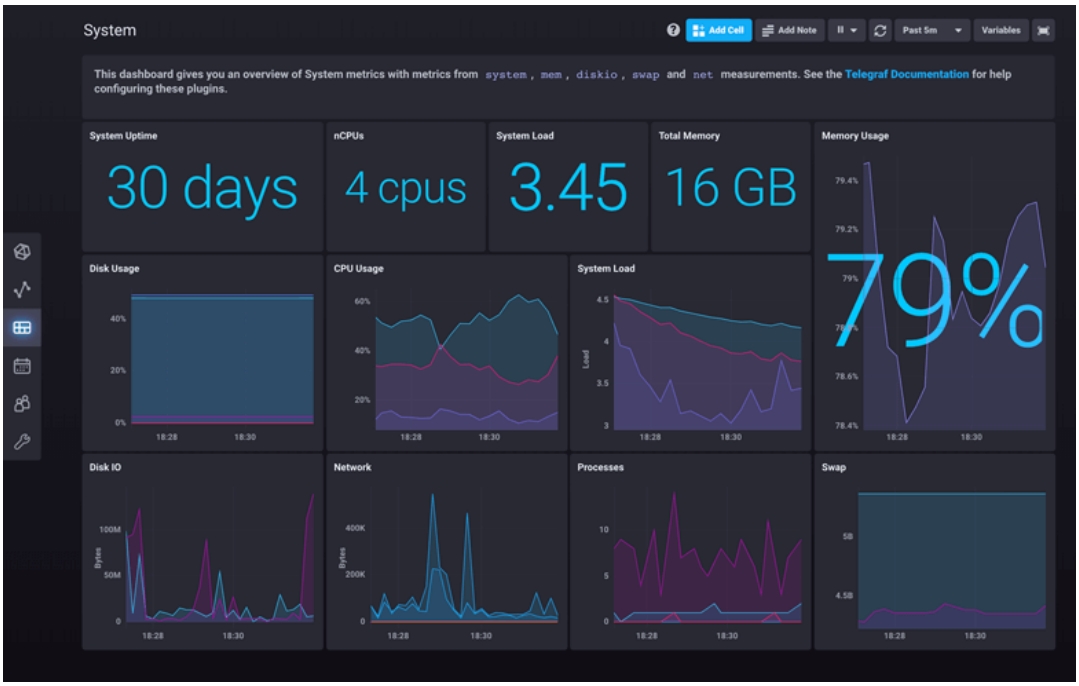


Figuur 11: InfluxDB push methode

Visualisaties

InfluxDB heeft sinds versie 2.x ingebouwde visualisatie mogelijkheden. Dit is in tegenstelling tot Prometheus inbegrepen in de database. Dit zorgt ervoor dat er geen aparte koppeling gemaakt hoeft te worden met een externe database om deze in te laden voor de visualisaties.

De visualisaties die mogelijk zijn, zijn vergelijkbaar met die in Grafana. In figuur 12 is een voorbeeld te zien van een dashboard in InfluxDB.



Figuur 12: Dashboard InfluxDB

4.8.2. Conclusie

Na deze vergelijking is eruit gekomen dat er een voorkeur is voor de InfluxDB database. Het verschil tussen de databases is namelijk niet heel groot, alleen het gemak dat InfluxDB heeft zorgt voor een aanzienlijk voordeel.

Het verschil met Prometheus is namelijk dat InfluxDB al ingebouwde visualisaties heeft en goed samen werkt met Telegraf. Telegraf is onderdeel van hetzelfde bedrijf dat InfluxDB heeft ontwikkeld en daarom is de integratie tussen deze twee onderdelen erg goed. Dit zorgt ervoor dat de flow van data verzameling, dataopslag en visualisatie allemaal makkelijker geïmplementeerd kan worden. De visualisaties die de database mogelijk maakt is echter niet voor de eindgebruiker van toepassing, maar vergemakkelijkt het ontwikkelproces wel door meteen inzicht te geven in de data.

Omdat de InfluxDB database door middel van de Flux query taal en long-time query's i.c.m. bucket bewaartijd ervoor zorgen dat er aan requirements **M01, M02, M03, M04** en **S03** wordt voldaan is er gekozen om gebruik te maken van InfluxDB als database.

4.9. Hoe moet de sFlow data worden gevisualiseerd?

De database InfluxDB en Grafana hebben beide mogelijkheden om hun data te visualiseren. Deze visualisaties zijn in een dashboard te implementeren binnen deze programma's. Zo'n dashboard is handig om te gebruiken wanneer je daadwerkelijk bezig bent met de data.

Data verwerken en dashboards maken is handig als ontwikkelaar omdat je zo de data kan bewerken en verschillende type visualisaties kan gebruiken om deze data in te zien.

Voor de opdracht is het nodig dat niet alleen het NOC, maar ook de klanten hun VLAN-dataverkeer kunnen inzien. Het is daarom niet mogelijk om alleen het dashboard te gebruiken voor de visualisaties en verwerking van de data. De klanten wil je immers geen toegang geven tot het ingebouwde dashboard van de database aangezien hier data bewerkt kan worden.

Klanten van NDIX hebben sinds kort toegang tot een klantenportaal waar ze allerlei taken kunnen regelen. De opdrachtgever wil graag de data inzichtelijk maken in het klantenportaal. Hierdoor kunnen klanten voor hun eigen afgenomen VLAN's hun dataverkeer inzien.

4.9.1. Klantenportaal

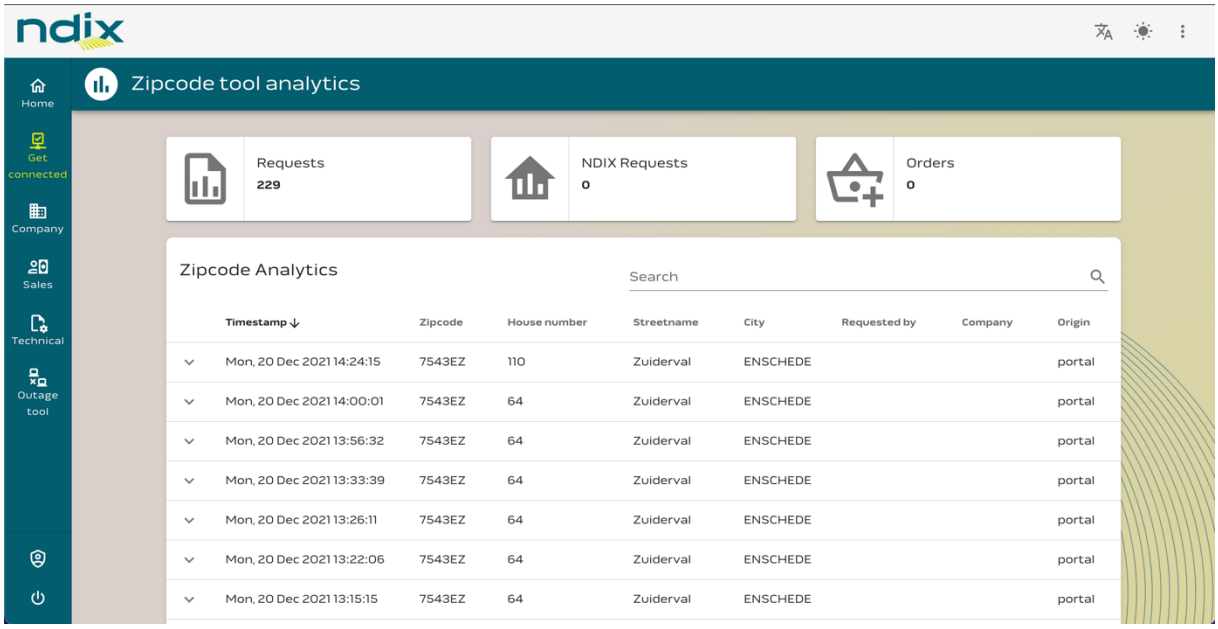
Het klantenportaal is een nieuw product van NDIX dat ze beschikbaar willen stellen aan hun klanten. In het klantenportaal kan de klant zelf allerlei zaken regelen die op dit moment nog handmatig gebeuren en erg tijdrovend zijn. Deze taken zijn bijvoorbeeld het aanvragen van VLAN's en het wijzigen van klantgegevens. Het aanvragen van VLAN's gebeurt op dit moment namelijk nog via de website van NDIX en wordt allemaal handmatig gecontroleerd door een NDIX-medewerker. Door de automatiseringen van het klantenportaal kan dit het NDIX-personeel veel tijd schelen en meer gaan dienen als een controlerende functie in plaats van uitvoerende functie.

Een van de requirements (**M09)** van de opdracht is dat de visualisaties zichtbaar zijn in het klantenportaal. Hierdoor zijn de visualisaties niet alleen voor het NOC intern beschikbaar, maar ook voor mensen van buitenaf zoals klanten.

Techniek
Het klantenportaal is gebouwd met verschillende technieken. Zo is de front-end gebouwd met VueJS samen met Vuetify en de back-end gebouwd met het PHP-framework Symfony. Een requirement van de opdracht is dat de visualisaties in het klantenportaal wordt geïntegreerd. De oplossing dient daarom ook te worden geschreven in deze talen/frameworks (**M10, M11)** om het compatibel te houden met het klantenportaal.

Functionaliteit
Het klantenportaal bestaat uit meerdere onderdelen, en ondanks de naam is het klantenportaal ook intern bij NDIX beschikbaar voor allerlei zaken. Zo kunnen de verschillende teams binnen NDIX andere functionaliteiten en opties tot hun beschikking hebben horende bij die afdeling. Zo kan het NOC bijvoorbeeld VLAN's goedkeuren en sales de gegevens wijziging aanvragen van klanten goedkeuren.

In afbeelding 13 is te zien hoe het klantenportaal eruitziet. Op deze pagina is de postcode-tool te zien.



Figuur 13: Klantenportaal

5. Werkwijze

Aan het begin van het afstuderen heeft de student in het plan van aanpak een strokenplanning gemaakt met de verwachtte taken en deadlines van de verschillende onderdelen van de opdracht. In deze strokenplanning staat aangegeven dat de student meerdere fases heeft van het afstuderen. Deze fases zijn: opstart, onderzoek, ontwerp, realisatie en oplevering.

Binnen het softwareteam van NDIX hanteren ze de scrum methode voor het ontwikkelen van software. Deze manier van werken wordt binnen het software domein vaak gebruikt om een (software)product te ontwikkelen.

5.1. Scrum

Scrum is een methode waarbij je door iteratief te werken een product gaat realiseren. Met iteratief werken wordt bedoeld dat je om de zoveel weken een evaluatie hebt van wat er gedaan is en zo nodig daarop bij de sturen. Tijdens deze weken wordt er door het team een selectie gemaakt van taken die gedaan moeten worden in deze periode. Deze periode met ontwikkeling heet een sprint.

Ondanks dat scrum een goed uitgedachte methodiek is, hanteert elk bedrijf het net weer wat anders. Zo varieert de sprint lengte per bedrijf en worden de rollen die binnen het scrummen worden gebruikt niet altijd vervult.

5.1.1. Rollen

Binnen scrum zijn er een paar vastgestelde rollen. Deze rollen zorgen ervoor dat de verantwoordelijkheid van bepaalde taken bij een persoon liggen. De rollen die een scrum team doorgaans heeft zijn: product owner, scrum master en het ontwikkel team.

Product owner
De product owner is de ‘eigenaar’ van het te maken product. Deze persoon geeft zijn wensen aan bij het ontwikkelteam en houdt tijdens de ontwikkeling van het product met een ‘birdseye view’ de voortgang in de gaten. De wensen van de product owner worden in user stories verwerkt zodat het ontwikkelteam hiermee bezig kan. Na afloop van elke sprint kijkt de product owner of de gemaakte functionaliteit overeenkomt met de visie die hij voor ogen had en stuurt deze indien nodig bij met nieuwe user stories.

De product owner is doorgaans een persoon die vanuit het management aangestuurd wordt. Ook zit de product owner niet in het ontwikkelteam. Dit is zo bedacht omdat anders de product owner tijdens het ontwikkelen van het product allerlei wijzigingen kan doen.

Ontwikkelteam
Het ontwikkelteam zorgt ervoor dat het product gerealiseerd wordt. Aan het begin van een sprint wordt er samen met de product owner bepaald wat voor user stories er gerealiseerd moeten worden. Een ontwikkelteam bevat verschillende personen met ieder zijn eigen expertise.

Doorgaans bevat het ontwikkelteam tussen de 3-9 personen die ervoor zorgen dat het product gerealiseerd kan worden. Daarnaast is het team tijdens de sprints zelfsturend dus zorgt deze er zelf voor dat de taken worden verdeeld en opgepakt.

Scrum master
De scrum master zorgt ervoor dat het scrum proces in goede banen verloopt. Zo faciliteert deze de daily stand-ups en zorgt deze ervoor dat iedereen zich aan zijn taken houdt. Denk bijvoorbeeld aan dat de product owner niet in het ontwikkelproces wordt meegenomen tijdens de sprint.

De scrum master is doorgaans een persoon die ervoor zorgt dat het scrum proces wordt gewaarborgd.

5.1.2. Sprints

Het eindproduct wordt gerealiseerd in meerdere sprints. Het gebruiken van sprints zorgt ervoor dat je in een iteratief proces het eindproduct realiseert. Binnen software is het gebruik van een iteratief proces verstandig omdat de product owner/klant tijdens het ontwikkelproces altijd nog met wijzingen komt.

Aan het begin van een sprint wordt er samen met het ontwikkelteam, scrum master en product owner gekeken naar de taken die gedaan moeten worden in de sprint. De afstudeeropdracht heeft verschillende requirements waar deze aan moet voldoen. Om deze requirements te verwerken in het scrum proces dienen deze worden opgesplitst in user stories. Deze user stories worden vervolgens in de backlog gezet waar deze opgepakt kunnen worden tijdens de sprints. Een user story is een onderdeel van een groter geheel wat opgesplitst is. Door requirements in user stories te verwerken wordt ook dit een iteratief proces.

5.2. NDIX

Bij NDIX hanteren ze sprints van twee weken. Aan het begin van de sprint wordt er met het ontwikkelteam besproken welke user stories er opgepakt moeten worden.

User stories en scrum poker
Aan deze user stories worden story points toegekend door middel van scrum poker. Elk van de ontwikkelaars geeft tegelijk aan hoeveel story point deze denkt nodig te hebben voor de desbetreffende user story. Vervolgens wordt er overlegd waarom het team denkt zoveel punten nodig te hebben. Mocht de user story een te hoog aantal punten krijgen wordt deze opgesplitst in meerdere user stories. Zodoende duurt een user story nooit te lang en kan de ontwikkelaar steeds user stories afstrepen van de backlog lijst.

Daily stand-ups
Dagelijks wordt er in de ochtend om 09:30 een (online) stand-up gehouden. Een daily stand-up is ook onderdeel van het scrum proces.

Tijdens een daily stand-up wordt er door elk teamlid besproken wat deze de vorige dag gedaan heeft, wat deze die dag gaat doen en of er nog tegen problemen gelopen is.

5.3. Afstuderen

Het afstuderen bestaat uit meerdere fases. Deze fases zijn door middel van een stroken planning ingedeeld. Deze strokenplanning staat alleen niet gelijk aan de twee wekelijkse sprints die NDIX hanteert. Tijdens het afstuderen is er daarom voor gekozen om in de verschillende fases niet met de sprints te werken maar wel mee te doen met de daily stand-ups zodat de voortgang dagelijks besproken kan worden.

Naast de dagelijkse bijpraat momentjes heeft de afstudeerder om de zoveel tijd een gesprek gehad met zijn bedrijfsbegeleider om in meer detail de voortgang te bespreken van zowel de opdracht en het verslag.

De afstudeerder heeft tijdens het ontwikkelproces gebruik gemaakt van de State-Gate(waterval) methode (State-Gate, sd). Deze methode bevat de volgende fases: specificeren, ontwerpen, programmeren, testen, accepteren en implementeren.

Tijdens het vooronderzoek is gespecificeerd wat de wensen van de opdrachtgever waren hoe het huidige systeem werkt. Na het vaststellen van de wensen is de applicatie ontworpen en gerealiseerd.

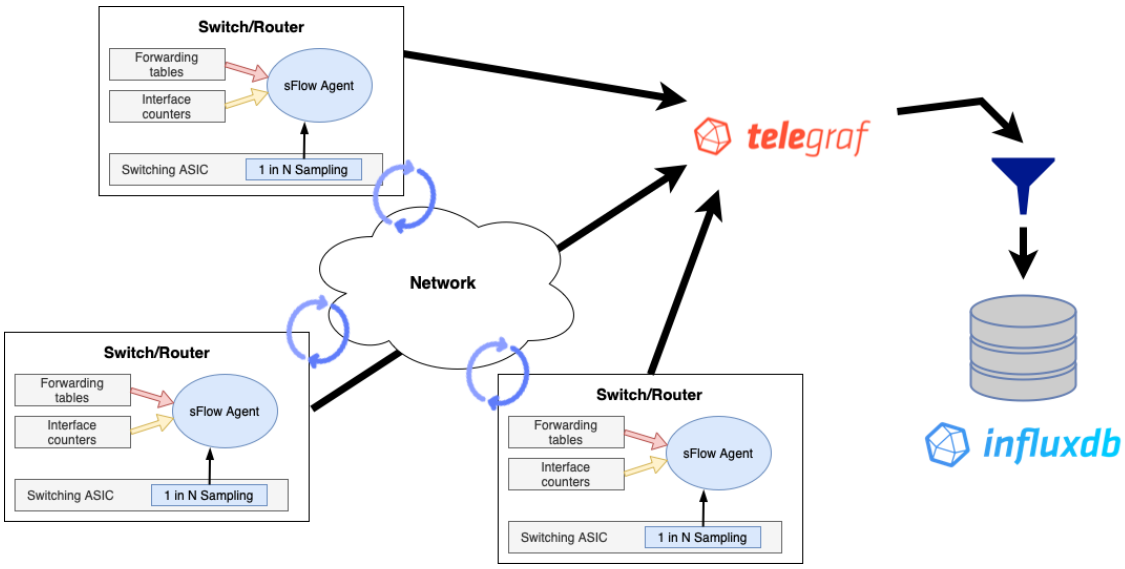
6. Ontwerp

In dit hoofdstuk wordt toelichting gegeven op de ontwerpen die zijn gemaakt binnen het afstudeerproject.

6.1. sFlow collector en opslag

De database waar de sFlow data in opgeslagen wordt is InfluxDB. Dit is geen relationele database maar een time series database. Deze database zorgt ervoor dat de data die ontvangen is bij de collector wordt opgeslagen in verschillende *buckets*, dat is bij InfluxDB de term voor database.

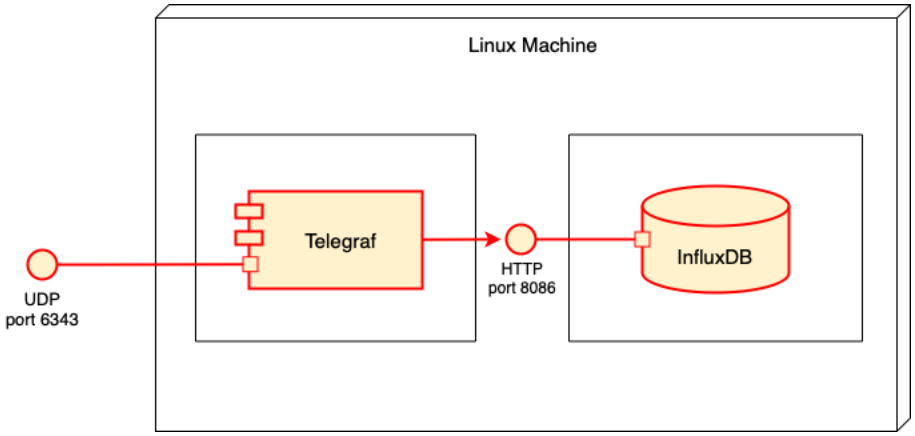
In figuur 14 is te zien hoe het sFlow protocol samen met Telegraf en InfluxDB fungeert. Op de switches van NDIX is sFlow al geïnstalleerd en geconfigureerd.



Figuur 14: sFlow collector en database

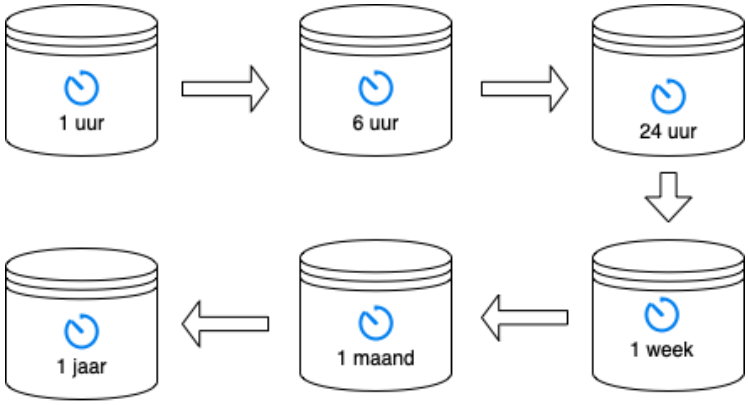
Op een server die aangeleverd is door NDIX zijn Telegraf en InfluxDB geïnstalleerd zodat de data gefilterd en opgeslagen kan worden (zie figuur 15).

Telegraf fungeert als sFlow collector en zorgt ervoor dat de ruwe pakketjes die binnenkomen op poort 6343 worden omgezet naar bruikbare data. Deze collector krijgt een constante stroom van ruwe data pakketjes binnen vanuit de NDIX-switches op het netwerk.



Figuur 15: Server components diagram

InfluxDB wordt gebruikt als database. Omdat de constante stroom van data van honderden switches kan resulteren in een opslagprobleem is ervoor gekozen om de data na een bepaalde periode te aggregeren en op te slaan in een andere database. De data wordt in de verschillende databases voor een bepaalde periode opgeslagen en vervolgens verwijderd.



Figuur 16: Verschillende databases met bewaartijden

In figuur 16 is te zien dat de data bewaard wordt voor een vastgestelde periode en vervolgens naar een andere database wordt weggeschreven. De laatste bucket waar data naar wordt heen geschreven heeft een bewaartijd van 1 jaar. Deze bucket zorgt ervoor dat er wordt voldaan aan requirement **S03.**

6.2. Klantenportaal

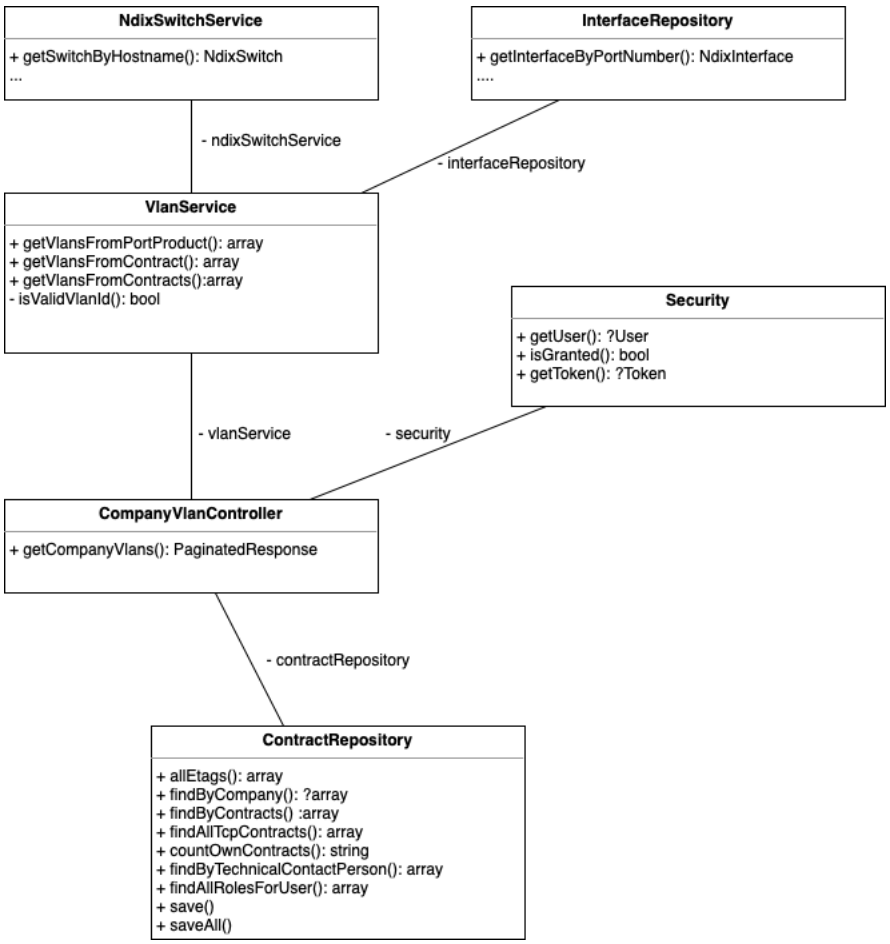
De onderdelen die gerealiseerd zijn tijdens het afstuderen zijn geïmplementeerd binnen het klantenportaal van NDIX. Dit klantenportaal bestaat uit een backend (NDIX API) en frontend.

6.2.1. Backend

De backend van het klantenportaal is een bestaande applicatie genaamd NDIX API. Het is een applicatie die is gebouwd door middel van het PHP-framework Symfony. Dit framework heeft van zichzelf een duidelijke structuur waar binnen het NDIX-project ook aan gehouden wordt.

Symfony is te gebruiken als een MVC (Model View Controller) framework, maar NDIX heeft ervoor gekozen om het framework alleen te gebruiken voor de API kant van het klantenportaal.

Om Symfony alleen als backend te gebruiken betekend dat het frontend gedeelte, wat standaard in het framework zit, niet wordt gebuikt. Alle routes die worden gebruikt hebben allemaal ***/api*** ervoor.



Figuur 17: Class diagram

In het class diagram, te zien in figuur 17, is te zien hoe de structuur in elkaar zit voor het ophalen van de VLAN's voor klanten. We willen namelijk vanuit de *CompanyVlanController* de VLAN's teruggeven die bij dat bedrijf horen.

Voor het ophalen van deze VLAN's moet er vanuit de *CompanyVlanController* eerst gekeken worden naar welke contracten dit bedrijf heeft. Wanneer de contracten bekend zijn kan er worden gekeken naar welke PortProducts daarbij horen. In een PortProduct zit namelijk de hostname van de switch en portnumber van de interface.

Met deze gegevens is het vervolgens mogelijk om de VLAN's op te halen. Dit gebeurt hier door het gebruik van de *vlan service* met de methode *getVlansFromPortProduct*.

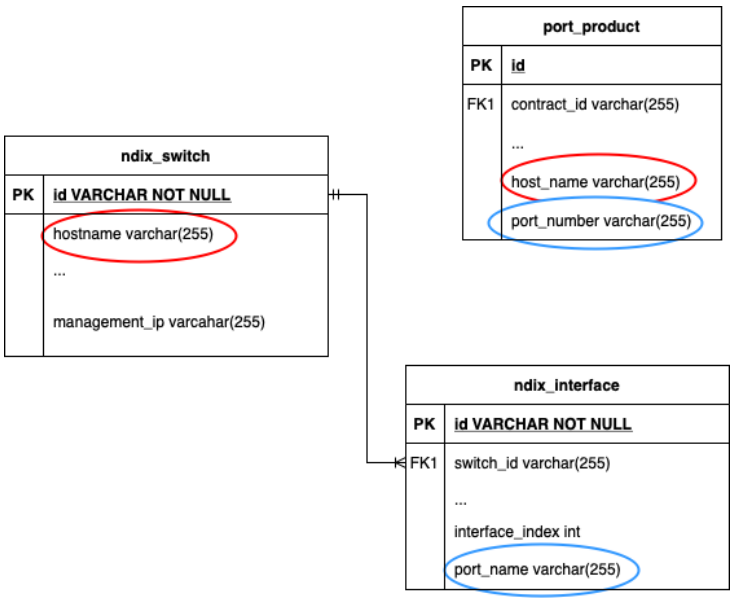
6.2.2. Database

In de huidige database van NDIX is er geen directe relatie tussen de gebruiker en zijn VLAN's. In de ERD (zie figuur 19) is te zien dat de gebruiker door middel van relaties van de **user** tabel naar de **port_product** tabel kan komen. In dit figuur zijn alleen de tabellen/kolommen te zien die van toepassing zijn voor het afstudeerproject.

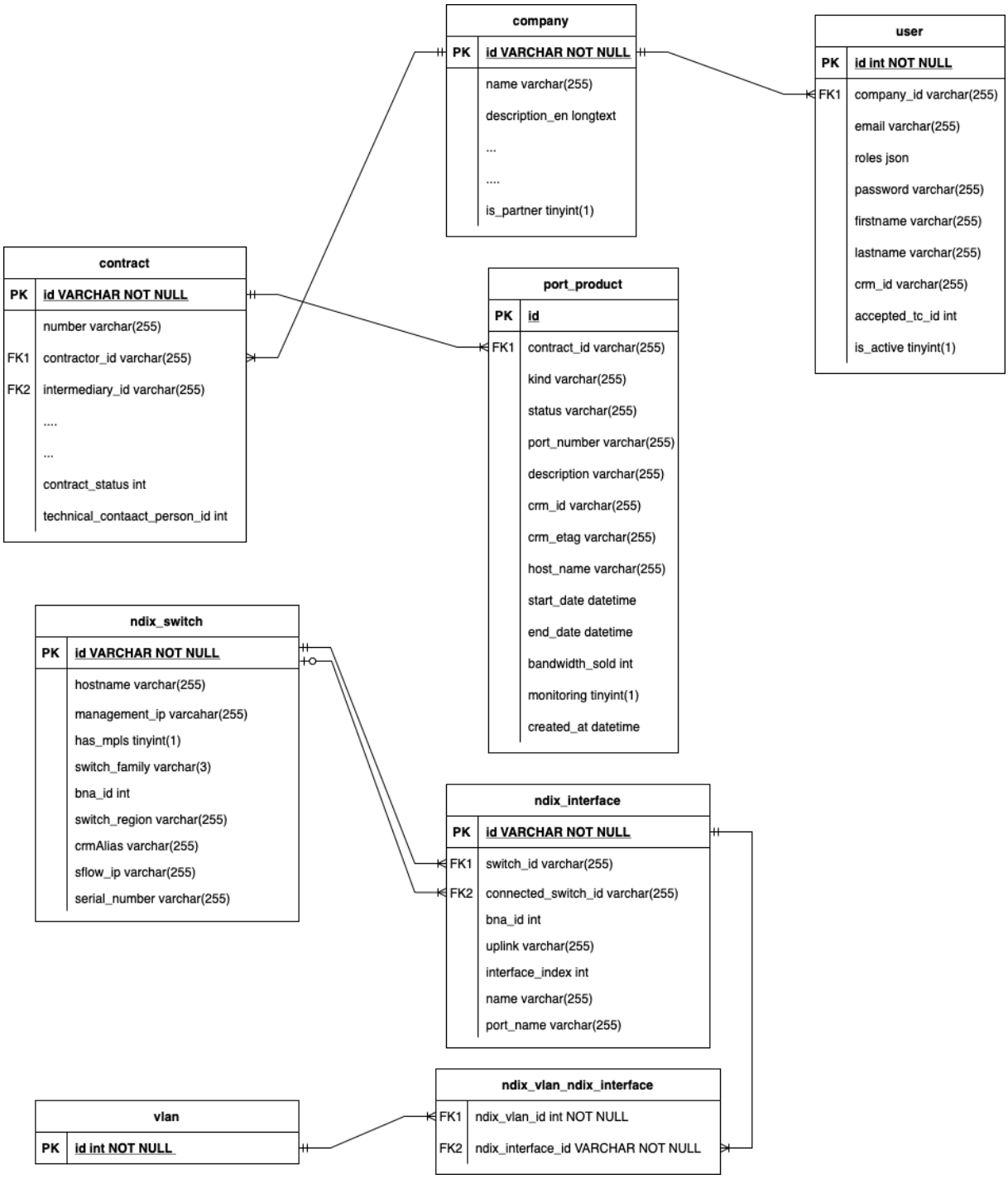
Om van een **user** naar een **port_product** te komen moeten er meerdere relaties worden bekeken. Een **user** bevat een *company_id*, elke **user** hoort namelijk bij één **company**. Met het *company_id* kan je vervolgens alle **contracten** ophalen. Elk **contract** is gekoppeld aan de **company** welke deze heeft afgenomen. Elk **contract** is gekoppeld aan een of meerdere **port_products**. En vanuit deze **port_products** is het mogelijk om de bijbehorende **VLAN's** op te halen.

Elke **VLAN** is gekoppeld aan een **ndix_interface**. Een **VLAN** kan op meerdere **ndix_interfaces** zitten, maar een **ndix_interface** kan ook meerdere **VLAN's** bevatten. Een **ndix_interface** behoort tot één **ndix_switch**.

In de **port_product** tabel staan de kolommen *host_name* en *port_number.* Door middel van deze kolommen is het mogelijk om bij de bijbehorende **ndix_switch** en **ndix_interface** te komen. In de database bestaat hier om technische redenen echter geen relatie tussen. Om dit op te lossen is er in het framework een relatie gemaakt tussen de verschillende classes.



Figuur 18: Missende relatie port_product



Figuur 19: ERD User to VLAN

6.2.3. Frontend

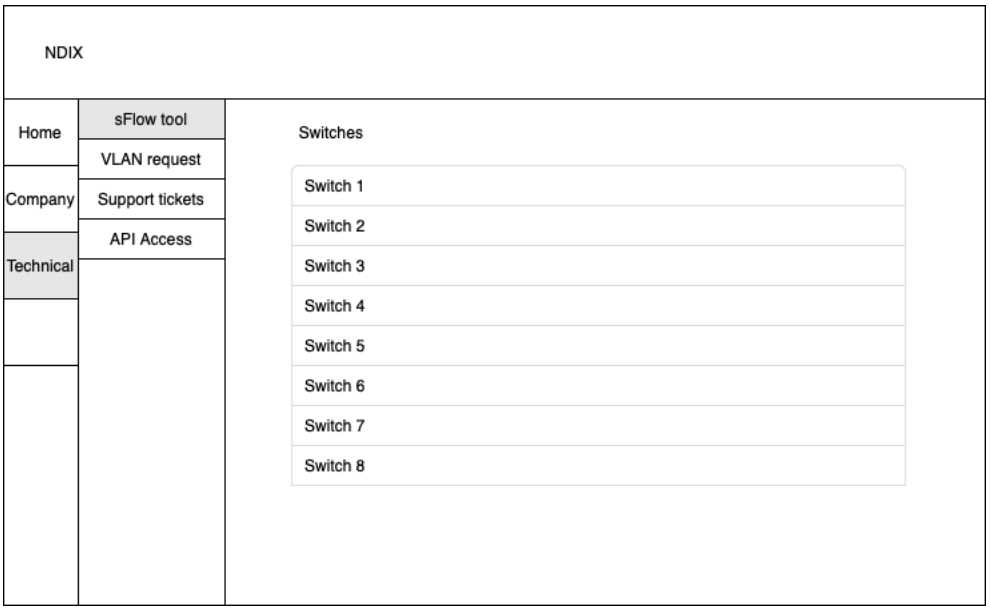
De onderdelen van het NOC en de klant moeten beide zichtbaar zijn binnen het klantenportaal.

Het NOC heeft toegang nodig tot de data van alle switches, interfaces en VLAN's. Deze data kan worden weergegeven onder de pagina *Technical*. Deze pagina is op dit moment alleen toegankelijk voor het NOC omdat hier technische functionaliteiten zijn bedoeld om intern gebruik van te maken.

6.2.3.1. Applicatie ontwerp

Switches

Het NOC moet allereerst een switch kunnen selecteren die het NDIX-netwerk bezit. In de database van het klantenportaal staan deze switches. In figuur 20 is een wireframe te zien van het klantenportaal met het overzicht van deze switches. De gebruiker kan hier een keuze maken tussen de verschillende switches.

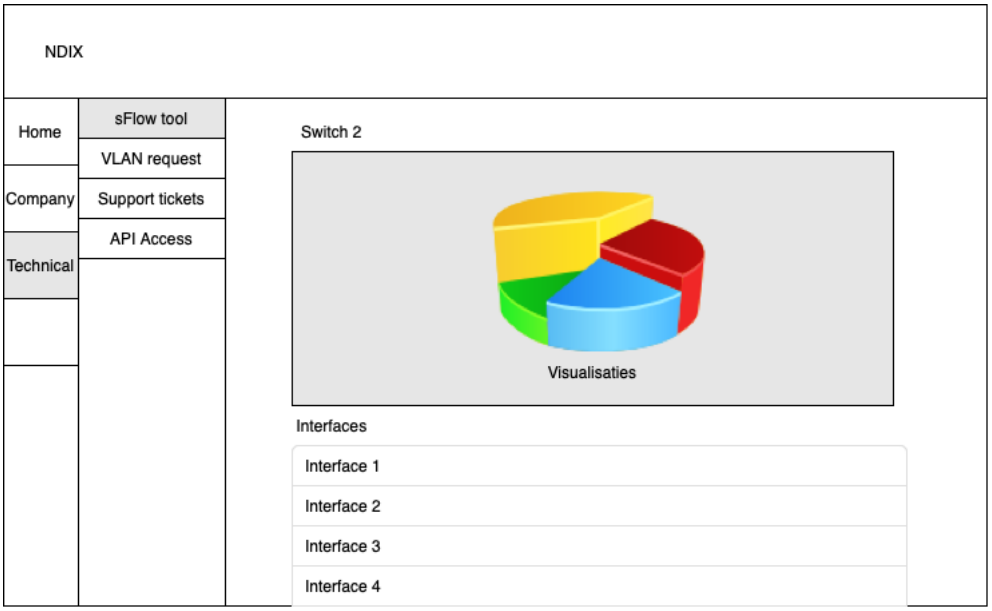


Figuur 20: Wireframe switches lijst

Switch

Wanneer de gebruiker een switch heeft geselecteerd kan deze van de gekozen switch het dataverkeer inzien. Deze data wordt opgevraagd bij de InfluxDB database. Er zijn verschillende visualisaties mogelijk omdat de InfluxDB database meerdere buckets heeft waar data staat opgeslagen voor verschillende tijdsperiodes.
Wanneer de gebruiker een switch heeft gekozen is het vervolgens mogelijk om te kiezen tussen de verschillende interfaces die bij deze switch horen. Een switch kan een of meerdere interfaces bevatten.

In figuur 21 is te zien dat de gebruiker de visualisaties van de switch kan inzien eb vervolgens een keuze kan maken uit verschillende interfaces die deze switch bezit.



Figuur 21: Wireframe switch met interfaces

Interface
Wanneer de gebruiker een interface selecteert komt deze vervolgens op een vergelijkbare pagina. Deze pagina geeft de visualisaties weer van het dataverkeer van een interface en geeft een lijst weer met de mogelijke VLAN's die deze interface bevat.

VLAN
Wanneer de gebruiker een VLAN geselecteerd heeft ziet deze alleen de visualisaties van deze VLAN. Op deze pagina is geen lijst te zien omdat onder een VLAN geen onderdelen meer vallen.

6.2.4. Component structuur

Een requirement (**M09**) die het bedrijf heeft gesteld is dat het dashboard binnen het huidige klantenportaal geïmplementeerd moet zijn. Dit betekent dat het dashboard wordt opgebouwd met de VueJS library welke ook gebruik maakt van de Vuetify library voor visuele componenten (**M13**).

De structuur van deze componenten is bepaald door het afstudeerbedrijf en wordt daarom ook overgenomen voor het dashboard. Het beginpunt van de applicatie is **index.html** waar het component wordt ingeladen vanuit ***index.js***. In dit bestand staan standaard routes zoals Home, Login en Register gedefinieerd. In dit bestand worden ook andere router bestand ingeladen.

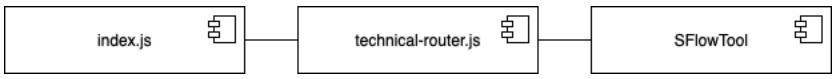
Voor de verschillende pagina's zijn dit ook andere bestanden; voor het technische NOC wordt de route in ***technical-router.js*** gedefinieerd en voor de klanten in ***company-router.js***.

Deze routes verwijzen door naar een Vue view-component met daaronder weer componenten die de webpagina de nodige informatie laat weergeven.

6.2.4.1. NOC

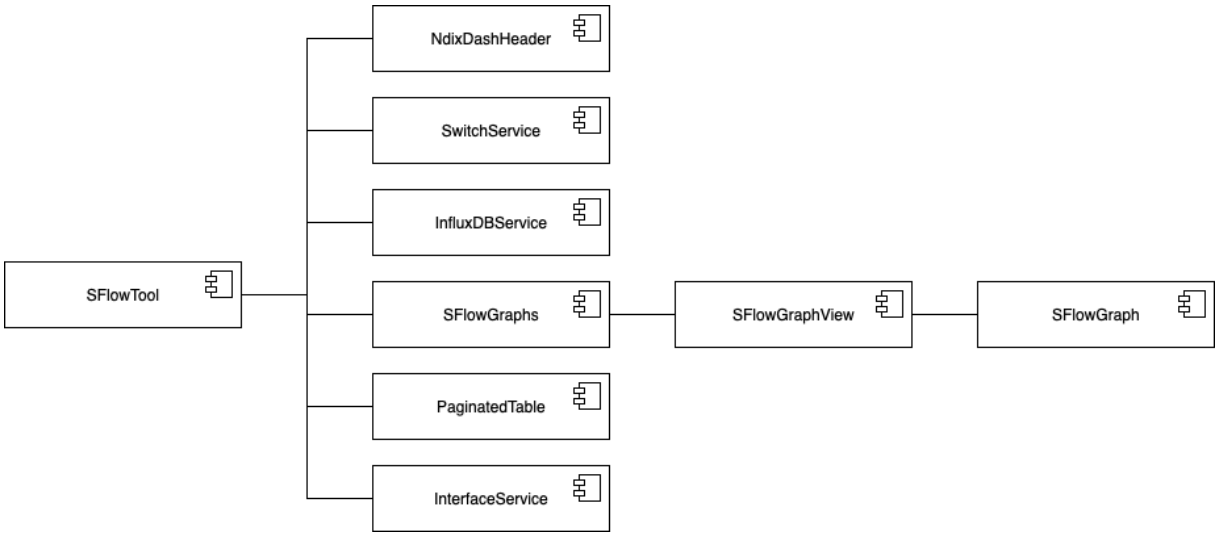
Het NOC heeft in de ***technical-router.js*** meerdere routes staat met technische doeleinden. Omdat de sFlow tool alleen voor intern gebruik voor het NOC is, wordt deze in de ***technical-router.js*** gezet.

In figuur 22 is te zien hoe deze Vue componenten zijn opgebouwd. SFlowTool is hier een view waar vervolgens componenten in worden geladen.



Figuur 22: NOC SFlowTool view

De view SFlowTool laadt zelf ook andere componenten in. De componenten die SFlowTool gebruikt is te zien in figuur 23.



Figuur 23: SFlowTool components

NdixDashHeader wordt gebruikt op nagenoeg elke pagina. Dit component zorgt ervoor dat de styling van de header op elke pagina hetzelfde is. In dit component kan je een icoon, titel en eventueel een zoekbalk zetten.

De SwitchService wordt gebruikt om de lijst met switches op te halen. Daarnaast wordt deze ook gebruikt om één specifieke switch op te halen met de bijhorende interfaces. De SwitchService verzorgt de methodes die vervolgens met de NDIX API communiceren.

De InfluxDBService heeft methodes waarmee deze met een package, die vanuit InfluxDB geleverd is, een connectie kan maken met de Influx database en daar de benodigde gegevens kan opvragen.

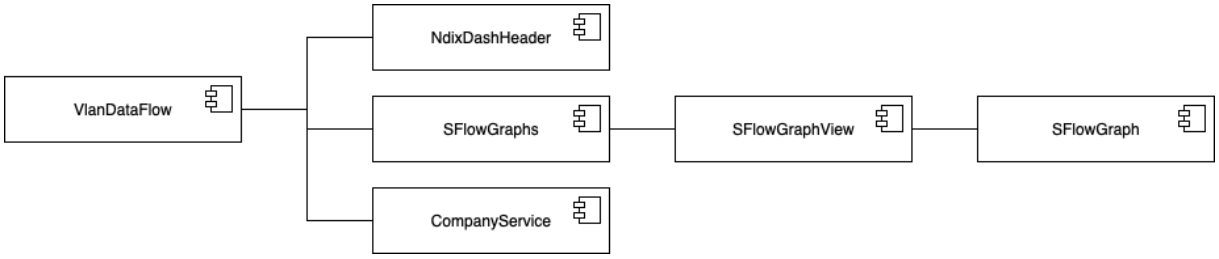
Het SFlowGraphs component geeft een accordeon weer met daarin de mogelijke opties voor data weergave. In deze dataweergaves die een uur, 6 uur, 24 uur, week, maand en jaar laten zien wordt een grafiek ingeladen doormiddel van het SFlowGraphView component. In dit component zit alleen nog de daadwerkelijke grafiek die ge-extend is van de Vue ChartJS package. Er is gekozen om dit component te laten extenden van BarChart.

6.2.4.2. Klant

Klanten hebben toegang tot hun eigen VLAN's en de daarbij horende grafieken. De klanten kunnen deze pagina benaderen via het menu-item "company". Onder dit menu-item staan meerdere relevante pagina's waar gebruikers hun bedrijfsgegevens kunnen inzien, wijzigen en VLAN's kunnen aanvragen. Hierbij komt dus ook de VlanDataFlow te staan. De VlanDataFlow wordt toegevoegd aan de ***company-router.js*** waar deze ingeladen kan worden wanneer klanten op de pagina **/company/vlans** komen. In figuur 24 is te zien hoe deze structuur van ***index.js*** tot ***VlanDataFlow*** is opgebouwd.

Figuur 24: Klant VlanDataFlow view

Het component VlanDataFlow fungeert als een view binnen het klantenportaal. Deze view laadt ook andere componenten in om de, bij de klant horende, VLAN's te kunnen opvragen en weer te geven.



Figuur 25: Component structuur VlanDataFlow

In figuur 25 is te zien hoe de componentenstructuur van de view eruitziet.

NdixDashHeader zorgt er net zoals bij de NOC-pagina voor dat de styling van het header menu er uniform uitziet zoals op elke pagina. Hier wordt dan ook een icoon en titel aan meegegeven.

Het SFlowGraphs component laat in een accordeon de mogelijke visualisaties zien voor de opgevraagde VLAN. Dit component wordt hetzelfde gebruikt als bij de implementatie van het NOC.

CompanyService zorgt ervoor dat de klant de juiste VLAN's te zien krijgt. Deze service maakt gebruik van de NDIX API waar deze de juiste VLAN's van terug krijgt.

7. Implementatie

7.1. Backend

De implementatie van de backend bestaat in dit project uit drie onderdelen; Telegraf, InfluxDB en de NDIX API.

7.1.1. Telegraf

Telegraf biedt een kant-en-klare sFlow plug-in. In tabel 5 is te zien welke tags en waardes er te verkrijgen zijn volgens de documentatie.

Tags	Waardes
agent_address	bytes
source_id_type	drops
input_ifindex	packets
output_ifindex	frame_length
sample_direction	header_size
header_protocol	ip_fragment_offset
ether_type	ip_header_length
src_ip	ip_total_length
src_port	ip_ttl
src_port_name	tcp_header_length
src_mac	tcp_window_size
src_vlan	udp_length
src_priority	ip_flags
src_mask_len	tcp_flags
dst_ip	
dst_port	
dst_port_name	
dst_mac	
dst_vlan	
dst_priority	
dst_mask_len	
next_hop	
ip_version	
ip_protocol	
ip_dscp	
ip_ecn	
tcp_urgent_pointer	

Tabel 5: Telegraf sFlow waardes

Bij het implementeren van deze sFlow plug-in is geconstateerd dat de documentatie van de plug-in niet klopt. De velden die van belang zijn voor de opdracht, namelijk de VLAN-velden (src_vlan en dst_vlan) zijn niet beschikbaar in deze plug-in. De code van deze plug-in is geschreven voor meerdere sFlow Flowsample variaties, maar niet voor de VLAN-data.

Om dit te achterhalen is eerst gekeken naar wat er daadwerkelijk bij de collector binnenkomt. Om dit inzichtelijk te krijgen is er een script geschreven die de pakketjes inzichtelijk maakt in Wireshark. Dit is een applicatie waar datapakketjes op het netwerk onderschept en gelezen kunnen worden.

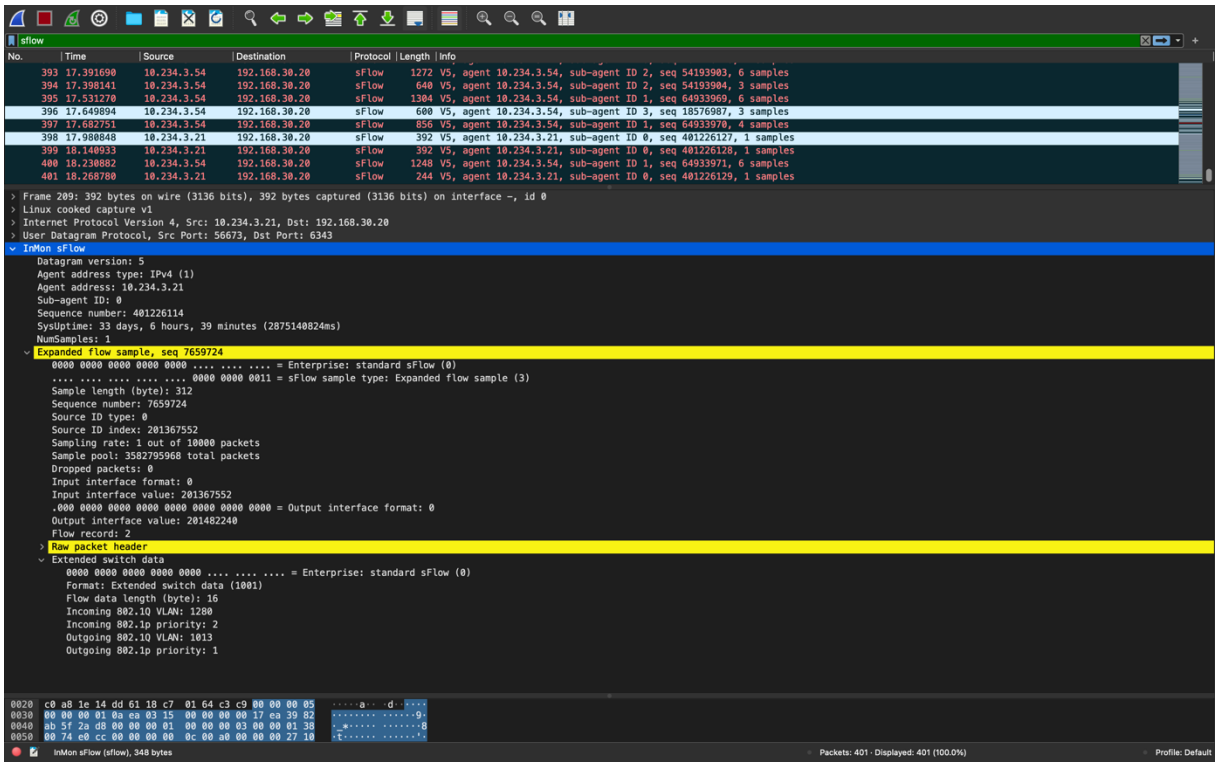
7.1.1.1. Wireshark

Het gemaakte script maakt via een SSH-tunnel connectie met de server en stuurt deze pakketjes door naar Wireshark. Op deze manier kan je de ruwe pakketjes die over het netwerk heen gaan onderscheppen en bekijken.

Het script wat dit uitvoert is als volgt:

```
ssh sascha@192.168.30.20 sudo -S tcpdump -i any -U -s0 -w - 'not port 22' | wireshark -k -i -
```

In figuur 26 is een schermopname te zien van Wireshark. Onder in deze afbeelding is te zien dat onder het kopje **Extended switch data** de Incoming 802.1Q VLAN en Outgoing 802.1Q VLAN te zien zijn. Deze data is nodig voor een compleet overzicht van het netwerk.



Figuur 26: Wireshark

In de schermopname zijn veel waardes te zien van het sFlow pakket. De data die nodig is voor het project is de switch IP, interface index en VLAN ID. In het onderschepte pakket zijn deze waardes inderdaad ook te zien. De switch IP is hier de *Agent address*, de interface index hier het *Input Interface value* en het VLAN ID is hier de *Outgoing 802.1Q VLAN.*

Om deze waardes alsnog te kunnen ontvangen is de plugin van sFlow aangepast op de lokale test server. Deze aanpassing zorgt ervoor dat de waardes src_vlan, src_priority, dst_vlan en dst_priority worden verwerkt door de collector (zie figuur 27 en 28).

```go
func (d *PacketDecoder) decodeExtendedSwitchFlowData(r io.Reader) (h ExtendedSwitchData, err error) {
    if err := read(r, &h.SourceVlan, "SourceVlan"); err != nil { return h, err }
    if err := read(r, &h.SourcePriority, "SourcePriority"); err != nil { return h, err }
    if err := read(r, &h.DestinationVlan, "DestinationVlan"); err != nil { return h, err }
    if err := read(r, &h.DestinationPriority, "DestinationPriority"); err != nil { return h, err }

    return h, err
}
```

Figuur 27: Nieuwe methode collector

```go
type ExtendedSwitchData struct {
    SourceVlan          uint32
    SourcePriority      uint32
    DestinationVlan     uint32
    DestinationPriority uint32
}

func (h ExtendedSwitchData) GetTags() map[string]string {
    return map[string]string{
        "src_vlan":     strconv.FormatUint(uint64(h.SourceVlan), base: 10),
        "src_priority": strconv.FormatUint(uint64(h.SourcePriority), base: 10),
        "dst_vlan":     strconv.FormatUint(uint64(h.DestinationVlan), base: 10),
        "dst_priority": strconv.FormatUint(uint64(h.DestinationPriority), base: 10),
    }
}
func (h ExtendedSwitchData) GetFields() map[string]interface{} {
    return map[string]interface{}{}
}
```

Figuur 28: VLAN en priority tags toevoegen

Na het toevoegen van deze methodes worden alle juiste waardes ontvangen. Echter omdat er zoveel tags met bijbehorende waardes zijn, worden deze eerst gefilterd in de Telegraf configuratie. In het ***telegraf.conf*** bestand wordt aangegeven welke waardes er wel en niet doorgestuurd moeten worden naar InfluxDB. In figuur 29 is te zien welke waardes hiervoor zijn ingesteld.

```
[[outputs.influxdb_v2]]
  ## The URLs of the InfluxDB cluster nodes.
  ##
  ## Multiple URLs can be specified for a single cluster, only
  ## urls will be written to each interval.
  ## urls exp: http://127.0.0.1:8086
  urls = ["http://192.168.30.20:8086"]

  ## Token for authentication.
  token = [redacted]

  organization = "NDIX-BV"
  bucket = "sflow_data"
  namepass = ["sflow"]
  fieldpass = ["bytes"]

  insecure_skip_verify = true
  taginclude = ["agent_address", "dst_vlan", "input_ifindex"]
```

Figuur 29: Telegraf configuratie

De bucket die ingesteld staat is de database waar de data heen wordt geschreven.

Namepass zorgt ervoor dat alleen de **sflow** metingen worden doorgelaten en fieldpass zorgt ervoor dat alleen de waarde **bytes** wordt doorgelaten.

Taginclude zorgt ervoor dat alleen de relevante tags bij de **bytes** waarde worden meegegeven. Dit zijn de tags die we uiteindelijk gebruiken om de inzichten te krijgen die we nodig hebben.

Telegraf is een constant draaiende applicatie die alle sFlow pakketjes ontvangt, filtert en doorstuurt.

7.1.2. InfluxDB

De database is geïnstalleerd op de server door middel van een docker container. Deze container is draaiende op poort 8086 en heeft een ingebouwde interface waar instellingen mee te regelen zijn.

Zo is het mogelijk om in deze interface een API-key aan te maken die nodig is in de Telegraf configuratie. Ook is het mogelijk om in deze interface buckets aan te maken. Voor de opdracht zijn er verschillende buckets aangemaakt met verschillende bewaartijden. In tabel 6 is te zien welke buckets zijn aangemaakt en welke bewaartijden elke bucket heeft.

Bucket naam	**Bewaartijd**
sflow_data	1 uur
sflow_data_6h	6 uur
sflow_data_24h	24 uur
sflow_data_7d	7 dagen
sflow_data_30d	30 dagen
sflow_data_1y	365 dagen

Tabel 6: InfluxDB buckets

De sflow_data bucket wordt direct gevuld vanuit Telegraf. Zoals aangegeven wordt hier alleen de waarde **bytes** opgeslagen met de tags agent_address, input_ifindex, en dst_vlan. Deze waardes worden voor een uur bewaard. Elke tien minuten wordt er een query uitgevoerd die deze data verkleind en weg schrijft naar de volgende bucket; sflow_data_6h.

Deze long-time query's zijn query's die om de zoveel tijd een gemiddelde berekenen van de data en deze wegschrijft naar de volgende bucket voordat de data verwijderd is. In figuur 30 is een voorbeeld te zien van deze query.

```
option task = {name: "sFlow downsize all -> 30s", every: 10m}

from(bucket: "sflow_data")
    |> range(start: -10m)
    |> filter(fn: (r) => r["_measurement"] == "sflow")
    |> filter(fn: (r) => r["_field"] == "bytes")
    |> aggregateWindow(every: 30s, fn: mean, createEmpty: false)
    |> map(fn: (r) => r)
    |> to(bucket: "sflow_data_6h")
```

Figuur 30: long-time query

In het laatste gedeelte van de query is een map() functie te zien. Deze functie is toegevoegd als tijdelijke oplossing voor een bug. Na het instellen van deze long-time query's als "tasks" werd opeens de data niet meer verwijderd; de bewaartijd werd niet gehandhaafd. De buckets hebben ieder een aangegeven bewaartijd, en na deze periode verwijderd de database deze data.

Tijdens het bekijken van de data die ontvangen werd in bucket sflow_data werd deze data netjes verwijderd na een uur tijd. Toen de tasks voor het aggregeren ingesteld waren, stopte de database opeens met het verwijderen van de data; de data bleef oneindig in deze bucket staan.

Na lang onderzocht te hebben waar het probleem zat bleek het een bug te zijn in de InfluxDB database. Wanneer er geen **map(fn: (r) => r)** functie gebruikt wordt aan het einde van deze task loopt de volledige verwijder gebeurtenis van de database vast. Tot op het moment van schrijven is hier nog geen permanente fix voor, maar deze tijdelijke oplossing zorgt dat ook de verwijder task van de database niet crasht.

Om de server niet te veel te belasten met intensieve tasks worden de query's vaker uitgevoerd dan de tijd dat de database de data opslaat. De task voor het aggregeren van de sflow_data bucket wordt namelijk elke 10 minuten uitgevoerd. Zo hoeft deze task niet het volledige uur in een keer te verwerken en wordt de server load verdeeld over meerdere momenten in het uur.

Task naam	Bucket	Draait elke
sFlow downsize all -> 30s	sflow_data → sflow_data_6h	10m
sFlow downsize 30s -> 2m	sflow_data_6h → sflow_data_24h	1h
sFlow downsize 2m -> 15m	sflow_data_24 → sflow_data_7d	3h
sFlow downsize 15m -> 4h	sflow_data_7d → sflow_data_30d	12h
sFlow downsize 4h -> 12h	sflow_data_30d → sflow_data_1y	24h

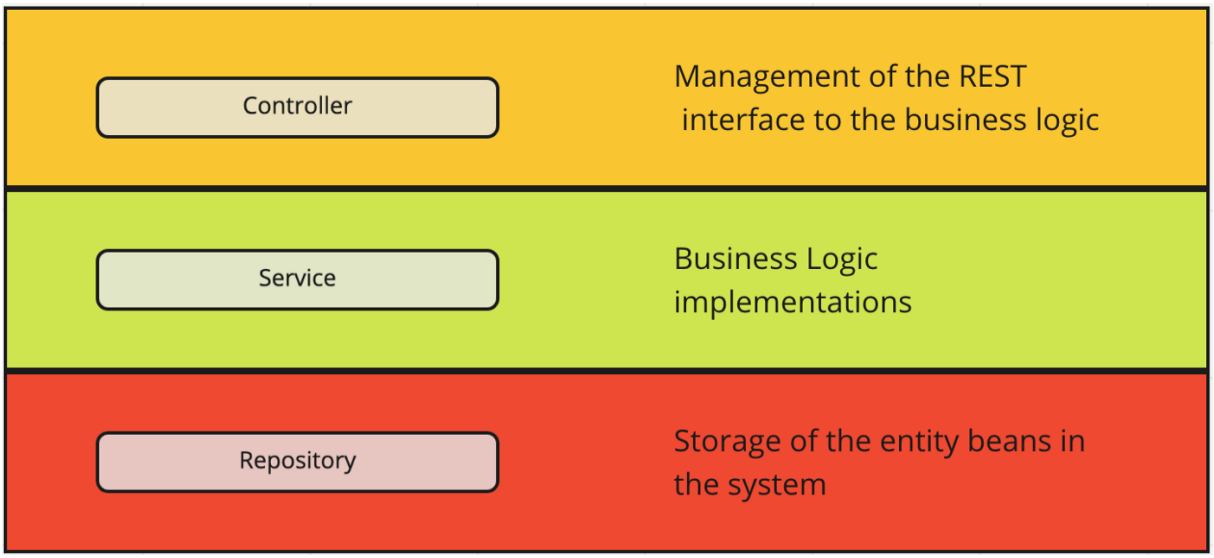
Tabel 7: InfluxDB tasks

In tabel 7 is te zien wanneer een bepaalde task wordt uitgevoerd en vanuit welke bucket deze data wordt opgehaald en geschreven.

7.1.3. NDIX API

De NDIX API is zodoende aangepast dat het mogelijk is dat de VLAN's opgehaald worden vanuit de company. Er is gekozen om de VLAN's op te halen vanuit het bedrijf omdat elke ingelogde gebruiker gekoppeld is aan één bedrijf. Gebruikers moeten namelijk hun eigen VLAN's terugkrijgen.

De NDIX API maakt binnen het PHP Symfony framework gebruik van het Controller-Service-Repository model. Dit model heeft de REST implementatie, business logica en database mutaties in verschillende bestanden opgedeeld.



Figuur 31: Controller Service Repository model (Collings, 2021)

Er is voor de implementatie een nieuwe controller aangemaakt genaamd *CompanyVlanController*. Deze controller zorgt ervoor dat de routes worden aangemaakt voor het opvragen van de VLAN's. In het framework worden de routes aangegeven door gebruik te maken van annotaties. In figuur 32 is te zien dat de class *CompanyVlanController* de route **/api/v1/company/vlan** heeft. Alle REST methodes die worden aangemaakt in deze class vallen onder deze route. Deze route is voor deze class dus de basis route.

```
/**
 * Class CompanyVlanController.
 *
 * @Route("/api/v1/company/vlan", name="company_vlan_")
 */
```

Figuur 32: Class annotatie

In het framework zit ingebouwde autorisatie die ervoor zorgt dat routes kunnen worden afgeschermd. In het ***security.yaml*** bestand staat aangegeven welke routes door welke rollen toegankelijk zijn. In figuur 33 is te zien dat de route van de *CompanyVlanController* class toegankelijk is voor de eindgebruiker en het NOC. Het gebruik van deze beveiliging is conform de bedrijfstandaard (**M12**).

```
- { path: '^/api/v1/company/vlan', roles: [ ROLE_NDIX_CUSTOMER, ROLE_NDIX_NOC ] }
```

Figuur 33: Security route

In de *CompanyVlanController* class is de *getCompanyVlans* methode toegevoegd. Deze methode is verantwoordelijk voor het terugsturen van de correcte VLAN's.
De methode zal eerst de huidige gebruiker opvragen welke bekend is vanuit de huidige sessie. De API-calls worden namelijk gemaakt met een token als authenticatie. Wanneer de gebruiker is gevonden wordt er gekeken naar welk bedrijf er bij die gebruiker hoort. Wanneer de gebruiker en het bedrijf bekend zijn wordt er in de *contract repository* gekeken naar de contracten die bij het bedrijf of de gebruiker hoort. Er wordt vanuit deze methode een andere methode aangeroepen in een repository class. Dit is vanuit het design ontwerp niet de standaard, maar omdat vanuit NDIX dit onderdeel van de codebase nog niet omgeschreven is naar dit model, wordt deze methode nu vanuit de controller aangeroepen.

De methode roept de *vlan service* aan om met de contracten te kijken naar de bijbehorende VLAN's. Aan deze methode worden de gevonden contracten gegeven en worden de bijbehorende VLAN's teruggegeven. Er wordt vanuit de contracten gekeken naar de VLAN's omdat port products bij een contract horen. Vervolgens kan je via de port products een VLAN koppelen.

```php
/**
 * @throws EntityNotFoundException
 * @throws NonUniqueResultException
 */
public function getVlansFromPortProduct(PortProduct $portProduct): array
{
    $switch = $this->switchService->getSwitchByHostname($portProduct->getHostName());

    $port = explode( separator: '-', $portProduct->getPortNumber())[0];
    $interface = $this->interfaceRepository->getInterfaceByPortNumber($switch->getId(), $port);

    if (null === $interface) {
        throw new EntityNotFoundException( entityType: NdixInterface::ENTITY_NAME,  propertyValue: $switch->getId().'-'.$port,  propertyType: 'switch-port');
    }

    return array_values($interface->getVlans()->filter(function (NdixVlan $vlan) {
        return $this->isValidVlanId($vlan->getId());
    })->toArray());
}

/**
 * @throws NonUniqueResultException
 * @throws EntityNotFoundException
 */
public function getVlansFromContract(Contract $contract): array
{
    $vlans = [];

    foreach ($contract->getPortProducts() as $portProduct) {
        $vlans = array_merge($vlans, $this->getVlansFromPortProduct($portProduct));
    }

    return $vlans;
}
```

Figuur 34: Implementatie contract naar VLAN

De *getCompanyVlans* methode kan vanuit het bedrijf de bijbehorende contracten opvragen. De contracten zijn op te vragen vanuit de *contract repository*.

Nadat de contracten ontvangen zijn worden de bijbehorende *port products* opgehaald. Bij het afsluiten van een contract kunnen daar één of meerdere *port products* bij horen. In deze *port products* staat de hostnaam van de switch en de interface index die nodig zijn om de VLAN's op te halen. Het ophalen van deze *port products* gebeurt in een loop door alle contracten heen.

In de database is er geen relatie tussen *port product* en de bijbehorende *switch* en *interface*. Dit wordt in de *getVlansFromPortProduct* afgevangen door te kijken naar de bijbehorende switch en interface (zie figuur 34). Eerst wordt de switch opgevraagd welke gekoppeld zit aan de port product. Deze vergelijking wordt gemaakt met de hostname die het *port product* bevat.

Na het ophalen van de switch wordt met de switch ID gekeken naar de interface met het opgevraagde port number. Het port number wordt voorafgaand eerst nog gesplit op `-` omdat in de database extra informatie kan staan voor externe partijen. Deze informatie is alleen niet belangrijk voor het opvragen voor van de VLAN's dus wordt deze hier uitgefilterd.

Wanneer er geen interface gevonden kan worden die bij de *port product* hoort wordt er een melding teruggestuurd die aangeeft dat de entiteit niet bestaat. Deze melding bevat de switch ID en port number.

Wanneer er wel een interface is gevonden worden de bijbehorende VLAN's opgehaald. Deze VLAN's worden opgehaald door middel van de relatie die gedefinieerd is in het framework. De interface entiteit bevat een veel-op-veel relatie met de VLAN's (zie figuur 35). De gedefinieerde veel-op-veel relatie zorgt voor een intersectietabel in de database genaamd *ndix_vlan_ndix_interface*.

Naast de intersectietabel zorgt het framework voor de gehele database structuur. Zo worden entiteit classes aangemaakt waar verschillende variabelen in staan die een tabel in de database representeren. Het framework maakt vervolgens op basis van de variabelen en annotatie informatie de tabellen en kolommen aan.

```
/**
 * @ORM\ManyToMany (targetEntity=NdixVlan::class, mappedBy="interfaces")
 *
 * @var Collection<NdixVlan> - A list of all vlans that this interface is part of
 */
private Collection $vlans;
```

Figuur 35: Entiteit relatie

Wanneer de gebruiker een API-call maakt om zijn VLAN's op te halen wordt alleen het switch IP, interface index en VLAN ID teruggestuurd. Alle andere informatie die de database tabellen bevatten is niet relevant voor deze API-call.

De informatie die wordt teruggestuurd wordt bepaald door middel van annotaties die worden gedefinieerd in de entiteit. Bij het terugsturen van de data wordt een groep meegegeven die bij alle bijbehorende entiteiten nagaat welke variabele wel of niet moet worden meegegeven. Omdat voor het ophalen van de data uit de InfluxDB database alleen de bovengenoemde velden nodig zijn, vallen deze variabelen onder de groep *VlanInfo*.

7.2. Frontend

Het klantenportaal van NDIX heeft voor het NOC en voor de klanten aanpassingen gekregen. Zo heeft het NOC onder de pagina *Technical* de sFlow Tool beschikbaar en kunnen klanten onder de *Company* pagina terecht voor hun VLAN-dataverkeer.

7.2.1. NOC

Voor het NOC is een nieuwe pagina aangemaakt waar het dataverkeer te zien is voor switches, interfaces en VLAN's. Deze pagina is opgebouwd uit verschillende componenten zoals te lezen in hoofdstuk 6.2.4.1.

De pagina is te benaderen vanuit het standaard slide-out menu wat het klantenportaal bevat. De SFlowTool pagina valt onder het kopje *Technical*, dit is een kop waar enkele technische pagina's onder staan. Deze pagina's zijn alleen toegankelijk voor het NOC.

URL
De URL van de pagina is opgebouwd uit de verschillende onderdelen waar je data van kan inzien. De URL is zo opgebouwd dat je steeds specifieker het dataverkeer kan inzien. De opbouw van de URL is als volgt:

/TECHNICAL/SFLOW-TOOL/SWITCH IP/INTERFACE INDEX/VLAN ID

- */* – Overzicht van alle switches op het NDIX-netwerk.
- */10.234.12.34* – Inzicht in het dataverkeer van de geselecteerde switch & lijst van interfaces die bij deze switch horen.

- */10.234.12.34/2012867* – Inzicht in het dataverkeer van de geselecteerd interface op deze switch & lijst van VLAN's die bij deze interface horen.
- */10.234.12.34/2012867/32* – Inzicht in het dataverkeer van de geselecteerde VLAN op deze interface en switch.

Er is gekozen om de URL zodoende op te bouwen zodat het NOC makkelijk naar de juiste pagina kan navigeren, ook zonder dat ze de frontend in het klantenportaal gebruiken. Het NOC heeft namelijk vaak de IP-adressen, interface indexen en VLAN ID's voor zich en hoeven daarvoor niet door de interface door te navigeren. Door in de URL de juiste parameters mee te geven belanden ze op de juiste pagina.

Structuur

De pagina SFlowTool laadt componenten in op basis van de URL. Deze componenten laden ieder andere data in. Op het moment dat de pagina geladen is, wordt er gekeken naar de variabelen die op dat moment gevuld zijn. Deze variabelen worden uit de URL gehaald en in de state gezet van het component.

Wanneer geen van de drie variabelen gevuld zijn, laadt de pagina een component in waar een lijst van alle switches te zien is. Deze switches worden opgehaald door de *switch service*. Deze service is verantwoordelijk voor de connectie met de NDIX API en bevat de methodes die ervoor zorgen dat switch data kan worden opgehaald, toegevoegd, gewijzigd en verwijderd.

Omdat NDIX veel switches heeft worden deze opgehaald in segmenten. De hoeveelheid switches die moeten worden teruggegeven per segment wordt meegegeven vanuit de tabel die wordt gebruikt als lijstweergave. Het component wat daarvoor zorgt is de *PaginatedTable*. Dit is een custom component die bestaat in de frontend om gesegmenteerde data weer te geven.

```
<PaginatedTable
  v-if="!queryData.switch_ip"
  :headers="switchHeaders"
  :items="switches.data"
  :loading="!switches.loaded"
  :item-count="switches.count"
  :row-callback="selectSwitch"
  @get-items="getSwitches"
/>
```

Figuur 36: PaginatedTable component

Het component, welke te zien is in figuur 36, wordt alleen ingeladen wanneer de variabele *queryData.switch_ip* nog geen data bevat. Deze *queryData* wordt op basis van de URL gevuld. Door middel van de *getSwitches* methode worden alle items opgehaald. Deze methode kijkt naar de ingestelde 'items per pagina' zodat die hoeveelheid wordt weergegeven.

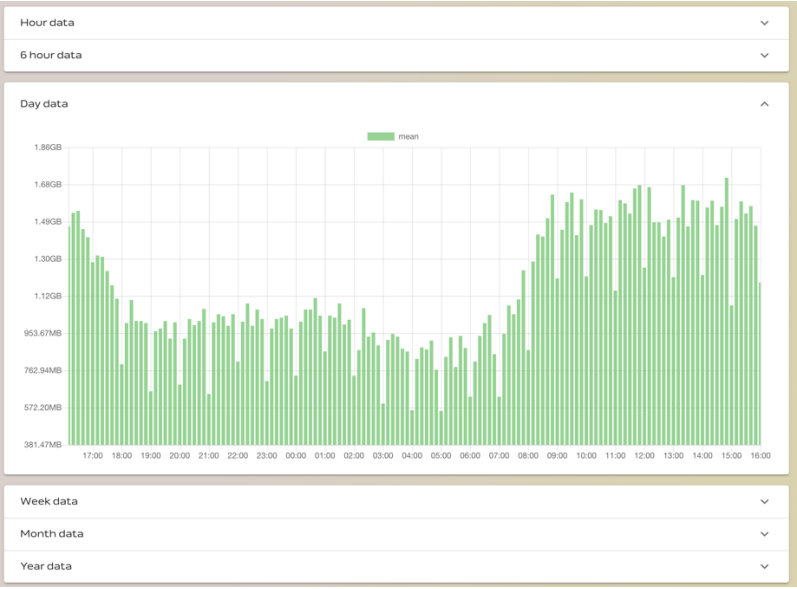
Wanneer een switch wordt geselecteerd wordt de switch IP naar de route gepusht. De pagina heeft constant een listner aanstaan voor wijzigingen in de route. Bij het pushen van deze nieuwe route wordt de *switch_ip* variabele gevuld en worden er andere componenten weergegeven op de pagina.

Weergave

Na het selecteren van een switch wordt er een accordeon component getoond waar de verschillende grafieken op te vragen zijn. De verschillende grafieken zijn op te vragen voor:

- 1 uur
- 6 uur
- Dag
- Week
- Maand
- Jaar

Wanneer de gebruiker één van deze opties selecteert wordt de grafiek ingeladen met de bijbehorende data. In figuur is 37 is te zien hoe de accordeon weergegeven wordt.



Figuur 37: sFlow data accordeon

De grafiek krijgt zijn data uit de *InfluxDB service*. Deze service is verantwoordelijk voor alle communicatie met de InfluxDB database. De datapunten die ingeladen worden zijn de gemiddeldes die opgeslagen zijn in de database. Omdat de query's voor het ophalen van de data redelijk intensief zijn voor de database, vooral wanneer je een jaar aan data opvraagt, wordt de data voor elk van deze opties maar één keer ingeladen en opgeslagen in de huidige sessie. In figuur 37 is te zien dat het dataverkeer drukker wordt na ongeveer 07:30. Dit komt omdat dan de werkdag begint voor veel mensen en de switch daardoor meer dataverkeer moet verwerken.

InfluxDB Service

De *InfluxDB service* is generiek opgesteld en zorgt ervoor dat door middel van de meegegeven data er een andere query wordt verstuurd naar de database. De methode die het versturen van de data mogelijk maakt is *getFluxQuery*. Deze methode ontvangt *bucket*, *aggregateTime*, *duration* en *data* als variabelen.

Bucket bepaald uit welke bucket de data wordt opgevraagd. Omdat de data met verschillende bewaartijden in verschillende buckets zit moet dit worden aangegeven in de query.

AggregateTime zorgt ervoor dat de data die in de bucket staat correct wordt geaggregeerd. Deze waarde is hetzelfde als de waardes die worden gebruikt voor het aggregeren van de data in InfluxDB database.

Duration bepaald hoe ver de query terug moet kijken in de bucket voor de data. Deze waarde staat gelijk aan de bewaartijd die staat ingesteld voor de verschillende buckets. Zo wordt bij de methode *getSixHourData* 6 uur terug in de tijd gekeken voor de data.

Data is een object dat meegegeven wordt met de drie verschillende URL-variabelen. Zo bevat deze de *switch_ip*, *interface_index* en *vlan_id*.

De accordeon vraagt de data op bij de *InfluxDB service* door het aanroepen van een van de volgende methodes: *getHourData*, *getDixHourData*, *getDayData*, *getWeekData*, *getMonthData* en *getYearData*. Elk van deze methodes ontvangt het *data* object en geeft deze door aan de *getFluxQuery* methode. Daarnaast vult elk van deze methodes de andere drie variabelen op met de bij deze methode horende data. In figuur 38 is te zien dat de velden *bucket, aggregateTime* en *duration* worden gevuld door de methode en *data* wordt meegegeven aan de volgende methode.

```
/**
 * @param {Object} data
 */
getHourData: function(data) {
  return getFluxQuery( bucket: "sflow_data",  aggregateTime: "30s",  duration: "-1h", data);
},
```

Figuur 38: InfluxDB service methode

De InfluxDB database wordt vervolgend gevraagd naar de data. Het opvragen van deze data gebeurt met de **influxdb-client** package die beschikbaar is voor NodeJS. Deze package is ontwikkeld door InfluxDB zelf en biedt de mogelijkheid om Flux query's te versturen naar de database. Flux is een eigen query taal is ontwikkeld is door InfluxDB om data makkelijk te kunnen query-en.

```
`from(bucket: "${bucket}")
        |> range(start: ${duration})
        |> filter(fn: (r) => r["_measurement"] == "sflow")
        |> filter(fn: (r) => r["_field"] == "bytes")
        ` +
customFilter +
`|> aggregateWindow(every: ${aggregateTime}, fn: mean, createEmpty: false)
        |> group(columns: ["_time", "_field", "_measurement"], mode:"by")
        |> sum()`;
```

Figuur 39: FLUX query

In figuur 39 is de query te zien die naar de database verstuurd wordt. Hier worden de variabalen die vanuit de vorige methode zijn meegegeven gevuld en zorgt *customFilter* ervoor dat de juiste filters worden toegevoegd op basis van het *data* object.

Interface en VLAN

Op de switch pagina is onder de accordeon een lijst te zien met interfaces die bij deze switch horen. Deze lijst met interfaces wordt ook in een PaginatedTable component weergegeven. Dit component wordt zowel bij switches, interfaces en VLAN's gebruikt. Wanneer de gebruiker uit deze lijst van interfaces een interface selecteert wordt vervolgens de interface index aan de URL toegevoegd. Door deze verandering in de URL worden de variabelen uit de URL opnieuw bekeken en opgevraagd. Omdat de *queryData* variabele nu een *interface_index* bevat verdwijnt de switch accordeon en interface lijst en komt daar een interface accordeon en VLAN lijst voor in de plaats.

Het ophalen van de data gebeurt op dezelfde manier als op de switch pagina; er wordt door middel van de *InfluxDB service* een methode aangeroepen die de juiste data teruggeeft. Ook op de VLAN pagina wordt door middel van dezelfde methodes de data opgevraagd. Zo is het uiterlijk uniform en kunnen de componenten en methodes hergebruikt worden. Op de VLAN pagina is echter geen *PaginatedTable* component te vinden aangezien onder een VLAN geen andere items vallen.

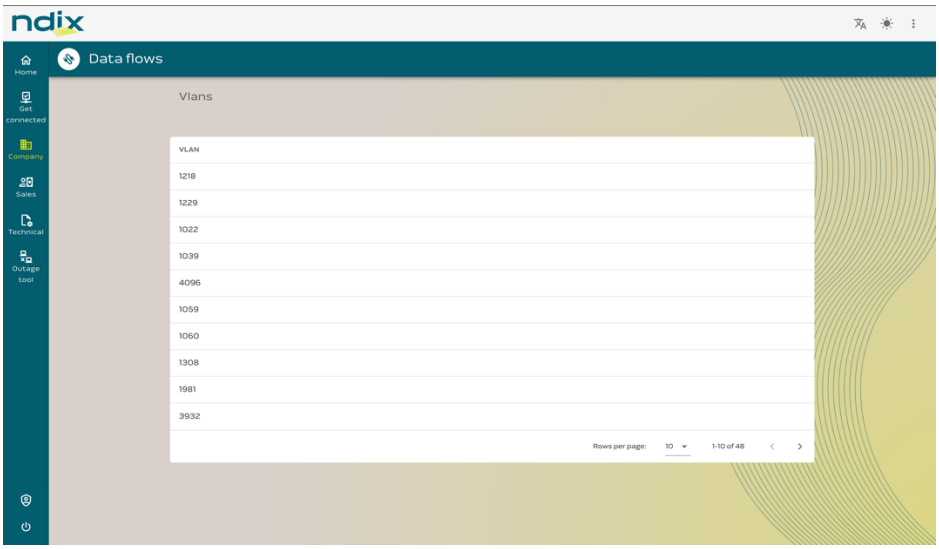
7.2.2. Klanten

Klanten van NDIX hoeven alleen data in te zien van hun eigen VLAN's. Deze VLAN's zijn op te halen via de company van een gebruiker.

In de NDIX API is het nodige aangepast zodat de gebruiker via zijn company de bijbehorende VLAN's kan opvragen. Deze API-call is benaderbaar vanuit de URL **/company/vlans**. Voor het uitvoeren van deze API-call is er in de *company service* een methode toegevoegd die dit uitvoert. De methode, genaamd *getVlansByCompany*, geeft de opgevraagde VLAN's terug.

De gebruikers kunnen hun eigen VLAN-data inzien op de pagina Data Flows. Deze pagina bevat een lijst met VLAN's die de klant bezit. De VLAN's worden weergegeven in een *DataTable* component. Dit component is afkomstig uit de Vuetify package. Het component zorgt ervoor dat er initieel maar 10 VLAN's worden getoond. Wanneer er meer VLAN's beschikbaar zijn kan de gebruiker de lijst vergroten. Dit type tabel is vergelijkbaar met het *PaginatedTable* component, alleen wordt er bij dit component door de *service* alle data al ingeladen waar de *PaginatedTable* alleen de data inlaadt die wordt opgevraagd.

In figuur 40 is de pagina te zien waar de klant zijn VLAN's kan inzien. Na het selecteren van een VLAN komt de klant op dezelfde pagina terecht als die het NOC heeft. Op deze pagina is dezelfde accordeon beschikbaar met de verschillende grafieken voor deze VLAN. Op deze manier kunnen klanten met weinig stappen makkelijk hun verschillende data inzien.



Figuur 40: Klanten VLAN overzicht

8. Voldoet de oplossing aan de eisen?

In dit hoofdstuk zal worden gekeken naar de requirements en of deze behaald zijn. Dit is een indicatie van of de afstudeeropdracht correct is opgeleverd. Er wordt per requirement en MoSCoW prioritering gekeken of deze behaald is.

8.1. Must

ID	Voldoet	Toelichting
M01	Ja	InfluxDB krijgt een constante stroom van sFlow data binnen. Deze data is in het dashboard inzichtelijk.
M02	Ja	Het systeem kan door middel van de InfluxDB API de VLAN-data opvragen.
M03	Ja	Het systeem kan door middel van de InfluxDB API de switch data opvragen.
M04	Ja	Het systeem kan door middel van de InfluxDB API de interface data opvragen.
M05	Ja	De gebruiker kan op de pagina “Data flows” zijn eigen VLAN’s inzien met het daarbij horende dataverkeer.
M06	Ja	Het NOC kan bij de sFlowTool per switch het dataverkeer inzien.
M07	Ja	Het NOC kan bij de sFlowTool per interface het dataverkeer inzien.
M08	Ja	Het NOC kan bij de sFlowTool per VLAN het dataverkeer inzien.
M9	Ja	De “Data flows” pagina en de sFlowTool zijn beide in het klantenportaal geïmplementeerd.
M10	Ja	De frontend is geschreven in JavaScript m.b.v. VueJS.
M11	Ja	De backend is geschreven in PHP m.b.v. Symfony.
M12	Ja	In de front- en backend is gebruik gemaakt van de bedrijfsstandard van autorisatie.
M13	Ja	De frontend is ontwikkeld met het gebruik van bestaande of Vuetify componenten die al in de huisstijl waren.

Tabel 8: Overzicht voltooiing must requirements

8.2. Should

ID	Voldoet	Toelichting
S01	Nee	In de NDIX API ontbreken erg veel fixtures die gebruikt worden bij het testen. Het is daarom niet mogelijk om nu goed te testen of van een bedrijf de juiste VLAN's terugkomen. Het testen met PHPUnit voor nu niet mogelijk door het ontbreken van de fixtures. Samen met de bedrijfsbegeleider is ter controle gekeken naar de VLAN's van een bedrijf en die waren kloppend. Voor nu is dit voldoende, maar in de toekomst dient dit wel uitgebreider getest te worden.
S02	Ja	De NDIX API is zodoende aangepast dat bij het gebruik van de *CompanyVlanController* en de daarbij horende *services* en *repositories* het systeem apart gebruikt kan worden als de rest.
S03	Ja	De data wordt in verschillende buckets tot een jaar bewaard in de InfluxDB database.

Tabel 9: Overzicht voltooiing should requirements

8.3. Won't

ID	Voldoet	Toelichting
W01	Ja	Valt buiten de scope van de opdracht.
W02	Ja	Valt buiten de scope van de opdracht.
W03	Ja	Valt buiten de scope van de opdracht.

Tabel 10: Overzicht voltooiing won't requirements

8.4. Overzicht

In tabel <> staat een overzicht van de wel en niet behaalde requirements geclassificeerd met de MoSCoW prioritering.

	Must	**Should**	**Won't**	**Totaal**
Aantal requirements	13	3	3	19
Aantal voldaan	13	2	3	18
Aantal niet voldaan	0	1	0	1

Tabel 11: Overzicht voltooiing alle requirements

Zoals te zien in tabel 11 is er voldaan aan bijna alle eisen. Door het niet halen van één should requirement is er uiteindelijk voldaan aan 18/19 requirements. Het niet behalen van de test requirement komt door een tekort in de NDIX API en zal in de toekomst worden opgelost. Het is daarom voor nu geen probleem dat deze ontbreekt. Hieruit valt te concluderen dat de opdracht succesvol is afgerond.

9. Reflectie

In dit hoofdstuk zal een algemene reflectie worden gegeven over het verloop van het project. Daarnaast zal er worden gereflecteerd over de behaalde competenties van het afstuderen.

9.1. Algemeen

Het afstudeertraject is de afgelopen 8 maanden niet vlekkeloos verlopen. Tijdens de implementatie van de verschillende onderdelen is er tegen verschillende problemen aangelopen. Dit heeft ervoor gezorgd dat de afstudeerperiode met een kwartiel is verlengd.

De implementatie van de sFlow collector met het gebruik van Telegraf zou een redelijk makkelijk te gebruiken oplossing moeten zijn geweest. Echter was dit niet het geval. In de documentatie van de sFlow plug-in stond namelijk beschreven welke tags en bijbehorende waardes erop te halen vielen, maar dit kwam niet overeen met de data die binnenkwam in de database. Dit probleem heeft lang geduurd om op te lossen omdat het probleem zich in de broncode van Telegraf zelf bevond. De benodigde VLAN-id waardes werden immers genegeerd in de code. Hier is vervolgens een uitbreiding voor geschreven in de broncode van Telegraf zelf. De documentatie is dus niet correct en heeft ervoor gezorgd dat er vertraging is opgelopen in dit proces.

Ook tijdens het implementeren van de database is er tegen problemen aangelopen. De database ontvangt een constante stroom van data en moet daarom data aggregeren en in andere databases schrijven.
Na het implementeren van de long-time query's, die ervoor zorgen dat de data ge-aggregeert en weggeschreven wordt in andere databases, bleek dat de database de ingestelde verwijder handelingen niet meer uitvoerde. Dit resulteerde in een database die de constante datastroom opgeslagen hield en de database en zo snel in omvang groeide. Omdat deze oplossing gebruikt gaat worden met honderden switches is het opslagtechnisch niet mogelijk om deze data voor altijd te bewaren. Echter was er in de documentatie niks te vinden over waarom dit probleem zich voor deed. Na lang zoeken is er een oplossing gevonden waardoor de data alsnog verwijderd wordt na een aangegeven tijdsperiode. Omdat ook dit probleem niet te vinden was in de standaard documentatie of stackoverflow vragen heeft dit lang geduurd om op te lossen. Het probleem bleek achteraf een bug te zijn van InfluxDB zelf en zij hebben hier ook een workaround voor gevonden.

Gedurende de loop van het project is het werkproces niet altijd soepel verlopen. Omdat het project zelfstandig uitgevoerd is, bleek het lastig om aan het scrum-proces te houden. Juist omdat het zelfstandig is uitgevoerd is het makkelijker om daily stand-ups te missen en code te schrijven op één branch omdat hier geen andere medewerkers van afhankelijk zijn. Door het missen van deze daily stand-ups is het contact met de bedrijfsbegeleider ook minder geweest dan gewenst en is het Trello bord pas op het einde van de afstudeerperiode in gebruik genomen.

Naast de verschillende problemen die gespeeld hebben is er wel een goede basis neergezet voor het visualiseren van het NDIX-netwerk en is het afstuderen een leuke en leerzame tijd geweest.

9.2. Competenties

Voor deze afstudeeropdracht zijn de volgende drie competenties gekozen en behaald op niveau 3: analyseren, adviseren en realiseren.

De competentie analyseren is behaald door het uitvoeren van een requirementsanalyse samen met de bedrijfsbegeleider en het onderzoeken van het huidige systeem en gebruikte protocol. Daarnaast is er gekeken of de opdracht voldoet aan de requirements. Deze informatie is terug te vinden in hoofdstuk 4 en 8.

De competentie adviseren is behaald door het uitgebreide onderzoek wat is gedaan naar het sFlow protocol, de mogelijke collectoren en verschillende databases. Uit dit onderzoek is een weloverwogen keuze gemaakt waar de implementatie van het project op is gebaseerd. Deze informatie is terug te vinden in hoofdstuk: hoofdstuk 4 en 6.

De competentie realisatie is behaald door het implementeren van een extensie van de sFlow collector in Telegraf, de InfluxDB task query's, uitbreiding in de NDIX API en frontend aanpassingen in het klantenportaal. Deze informatie is terug te vinden in hoofdstuk 6 en 7.

10. Aanbevelingen

10.1. sFlow Counter Sample

Op dit moment worden er bij de sFlow collector alleen de sFlow Flowsamples gebruikt om de datastroom te verkennen. De huidige sFlow collector die gebruikt wordt, een plug-in van Telegraf, ondersteund dit andere type sFlow pakket namelijk niet.

In de huidige oplossing wordt er gekeken naar de bytes die binnenkomen op een gegeven switch, interface of VLAN. Het is echter niet mogelijk om nu in te zien hoeveel bytes er in/uit gaan. Deze data is wel terug te vinden in de counter samples van sFlow. Er is meer onderzoek nodig om te kijken of deze samples ook de nodige switch IP, interface index en VLAN ID kunnen inzien. Bij het vooronderzoek tijdens dit project zijn de VLAN ID's namelijk alleen teruggevonden bij de flow samples, specifieker bij de *Extended Switch Data*.

Het gebruik van counter samples en de bijbehorende bytes in/uit kan een nog beter inzicht geven in het netwerk, vandaar ook de aanbeveling om dit in de toekomst te onderzoeken.

10.2. InfluxDB connectie

In de huidige oplossing is ervoor gekozen om de InfluxDB database rechtstreekst vanuit het klantenportaal te benaderen. Dit zorgt voor een tussenstap minder bij het ophalen van de sFlow data. Hier zit alleen geen validatie van klanten tussen. Het zou daarom nu theoretisch mogelijk zijn voor klanten om data op te halen van andere VLAN's.

Dit is in de toekomst af te vangen door de connectie met de database vanuit het portal te laten verlopen. Er is dan vooraf een check mogelijk om te kijken of de opgevraagde VLAN's horen bij de, door de klant afgenomen, VLAN's. Er is hiervoor wel een grote wijziging nodig in de NDIX API, maar dit zou ervoor zorgen dat de data beter beschermd is. Hier is daarom een aanbeveling om dit in de nabije toekomst te implementeren.

11. Bronnen

Phaal, P., & Panchen, S. (sd). *Packet Sampling Basics*. Opgehaald van sflow.org: https://sflow.org/packetSamplingBasics/

InMon Corp. (sd). *Application Notes*. Opgehaald van sFlow: https://sflow.org/using_sflow/application_notes.php

Deri, L., Mainardi, S., & Fusco, F. (2012). tsdb: A Compressed Database for Time Series. In A. Pescapè, L. Salgarelli, & X. Dimitropoulos, *Traffic Monitoring and Analysis* (p. 174). Vienna: Spinger.

Wayner, P. (2021, January 15). *Database trends: The rise of the time-series database*. Opgehaald van VentureBeat: https://venturebeat.com/2021/01/15/database-trends-the-rise-of-the-time-series-database/

influxdata. (2022). *Flux Documentation*. Opgehaald van influxdata: https://docs.influxdata.com/flux/v0.x/

Collings, T. (2021, Augustus 10). *Controller-Service-Repository*. Opgehaald van Medium: https://tom-collings.medium.com/controller-service-repository-16e29a4684e5

van Vliet, H. (2008). Software Engineering: Principles and Practice. Chichester.

State-Gate. (sd). Opgehaald van projectmanagementsite.nl: https://projectmanagementsite.nl/stagegating-waterval-methode/#.YkhRvy8RrRY