Professionele Bachelor Toegepaste Informatica

System data collector

Berre Mertens

Promotoren:

Peter Fastré
David Parren

Level27
Hogeschool PXL Hasselt

Bachelorpaper Academiejaar 2018-2019

Professionele Bachelor Toegepaste Informatica

System data collector

Berre Mertens

Promotoren:

Peter Fastré
David Parren

Level27
Hogeschool PXL Hasselt

Bachelorpaper Academiejaar 2018-2019

Dankwoord

Dit is het begin van de scriptie ‘System Data Collector’. Ik heb deze scriptie geschreven als eindwerk voor de studie Toegepaste Informatica aan de Hogeschool PXL.

Als eerste wil ik heel graag mijn stagebedrijf Level27 bedanken om mij de kans te geven om er stage te lopen. Verder zou ik mijn technische begeleider Roeland Andelhofs waar ik altijd terecht kon met vragen. Door hem kon ik deze stage tot een goed einde brengen.

Verder bedank ik iedereen die mij op een of andere manier gesteund heeft bij het tot stand komen van dit eindwerk. Ik ben zeer blij met alle begeleiding en steun die ik gekregen heb.

Abstract

Bij Level27 draaien er veel servers waarop er telkens meerdere websites/applicaties draaien. Iedere website/applicatie is natuurlijk anders en sommige vragen meer resources. In het huidige monitoringsysteem wordt enkel de server op zich gemonitord (CPU,RAM,disk,…) en niet de websites of applicaties.

Dit project bestaat uit twee delen. Ten eerste wordt er onderzoek gedaan naar een opensourcemonitoringtool voor het monitoren van multiserviceproviderplatform. Om een overzicht te hebben van welke tools er momenteel op de markt zijn, wordt er een literatuurstudie gemaakt. Hier wordt beschreven wat de karakteristieken zijn van elke tool en welke data ze kunnen binnenhalen. Uit deze literatuurstudie worden dan twee tools gehaald waaruit een POC wordt gemaakt. Dit onderzoek kijkt vooral naar de schaalbaarheid, automatisatie- en integratiemogelijkheden van elke tool.

Ten slotte worden er twee monitoringsystemen opgezet voor het monitoren van de Level27 infrastructuur. Op de gekozen tool wordt een performantietest gedaan en ook wordt er gekeken welke impact de *exporters* hebben op de servers.

Inhoudsopgave

1 Inhoud

Lijst van gebruikte figuren

Lijst van gebruikte tabellen

Lijst van gebruikte afkortingen

WAL	Write-ahead-log
UUID	Universally unique identifier
REST	Representational state transfer
RAM	Random-access memory
CPU	Central Processing unit
API	Application programming interface
PHP-FPM	Hypertext Preprocessor FastCGI Process Manager
FastCGI	Fast Common Gateway Interface
MSSQL	Microsoft SQL Server
SQL	Structured Query Language
HTTP	Hypertext Transfer Protocol
JSON	JavaScript Object Notation

Inleiding

Dit eindwerk bestaat uit twee delen: een stageverslag over het opzetten van de monitoringsysteem en een onderzoek naar de verschillende tools die hiermee kunnen helpen.

Level27 heeft de stageopdracht uitgeschreven omdat ze een uitgebreider monitoringsysteem nodig hebben voor het monitoren van de websites en de applicaties. Het monitoringsysteem zal bestaan uit een monitoringtool gecombineerd met Grafana en de verschillende *exporters* voor het verschepen van de data naar de monitoringtool. De gebruikte *exporters* moeten data kunnen ophalen waardoor websites en applicaties op de servers kunnen worden gemonitord. Met dit monitoringsysteem zal Level27 kunnen zien welke websites of applicaties voor problemen zorgen.

Uiteindelijk zal er nog gezorgd worden voor de visualisatie aan de klant zodat deze kan zien hoe het met zijn website of applicatie gaat.

In het stageverslag wordt Level27 en de stageopdracht gedetailleerder beschreven. Ook wordt de uitwerking van het monitoringsysteem toegelicht.

In het onderzoeksdeel van dit rapport wordt nagegaan welke tool het beste is voor deze situatie. Hierbij kan men denken aan resources, monitoring methode, clustering, schaalbaarheid, integratiemogelijkheden,… Deze vragen worden beantwoord aan de hand van een literatuurstudie en een *prove of concept*. De tools worden met elkaar vergeleken en hieruit komt een aanbeveling.

I. Stageverslag

Het eerste deel van dit eindwerk gaat over de stage bij Level27. Om de stageopdracht in de juiste context te zien, wordt Level27 eerst voorgesteld. Vervolgens wordt de probleemstelling en het doel van het nieuwe systeem toegelicht. Als laatste wordt het ontwikkelingsproces omschreven en het finale resultaat getoond.

2 Bedrijfsvoorstelling

2.1 Situering

Level27 is een IT-bedrijf in Bilzen dat zich focust op hosting voor consumenten tot middelgrote ondernemingen. Level27 is het verhaal van Peter Fastré die op zevenentwintigjarige leeftijd de uitdaging aanging om de dienstverlening van hostingbedrijven in een steeds veranderde IT-wereld te optimaliseren. Na dertien jaar is Level27 dan ook geëvolueerd tot een bedrijfsverhaal dat geschreven wordt door een groep van toegewijde collega's die streven naar het leveren van betrouwbare, veilige en stabiele hostingdiensten op maat en daarbij de waarden van Level27 hoog in het vaandel dragen.

2.2 Diensten

Level27 is gespecialiseerd in hosting. Deze worden dan opgedeeld in 4 diensten: domeinen, *shared* hosting, serverhosting en *managed services*. Bij Level27 kunnen er domeinen worden gekocht en kunnen gekoppeld worden aan bestaande e-mailadressen. Ook kunnen er bestaande domeinen worden overgeplaatst naar Level27 om van het advies en de diensten van Level27 te genieten.

Als tweede dienst kan er gebruik gemaakt worden van de *shared* hosting pakketten van Level27. Hier kan de klant kiezen om zijn website of applicatie te laten draaien.

Als derde dienst kan er gebruik gemaakt worden van de serverhosting pakketten van Level27. Hier kan men een server huren per maand. Deze servers kan men eventueel later nog altijd upgraden of downgraden.

Als 4de dienst hebben ze ook nog *managed services.* Level27 geeft proactief, support en opvolging van de diensten die je bij hun aankoopt.

3 Voorstelling stageopdracht

3.1 Probleemstelling

Eén van de waardes van Level27 is proactiviteit. Een cruciale taak van een hostingbedrijf is *uptime* en performantie monitoren. Dit zit dan ook in het DNA van Level27's dagelijks werk. Ze monitoren constant via Slack, Nagios, Dashboards, New Relic,… Wat er ook gebeurt, zij zijn eerst op de hoogte en dan pas de klant. [1]

Met deze waarde in gedachten wil Level27 zijn monitoringsysteem uitbreiden. Momenteel maakt Level27 gebruik van een Elastic stack systeem met zelfgemaakte beats die de data *pusht* naar Elasticsearch. De data die wordt opgehaald bestaat uit basissysteem *metrics* waarmee er kan gezien worden dat er iets mis is met de server, maar niet de aanleiding van bijvoorbeeld een CPU piek. Met het nieuwe systeem willen ze al deze vraagtekens schrappen.

3.2 Doelstelling

De doelstelling van het project is het opzetten van een opensourcemonitoringsysteem dat meer gegevens verzamelt dan enkel de systeemperformantie (zoals geheugengebruik, processen, applicaties). Het monitoringsysteem zal dan ook de websites op de *shared hosting servers* monitoren en zo zien welke websites het meest belasten op welke periode. Hierna zal ook deze data moeten worden gevisualiseerd aan de hand van opensourcesoftware.

Ten eerste wordt er een monitoringsysteem opgezet aan de hand van Docker-containers met *persistent storage*. Zo worden de configuraties en de data van de containers behouden bij het opstarten en stoppen van de containers.

Het nieuwe monitoringsysteem zal een totale vervanging van het huidige monitoring zijn en zal geïntegreerd worden in de bestaande systemen van level27.

Verder wordt het monitoringsysteem nog geïntegreerd met het controlepaneel om de data naar de eindklant te visualiseren.

Verder worden er ook performantietesten op het gekozen monitoringsystemen dat word opgezet. Ook wordt er getest naar de performantie van de verschillende data *exporters*.

4 Uitwerking stageopdracht

4.1 Basisconcepten

Docker

Docker is een opensource tool gemaakt om applicaties gemakkelijk te creëren, deployen en *runnen* door gebruik te maken van containers. Containers zorgen ervoor dat een applicatie verpakt wordt met alle nodige bibliotheken en andere afhankelijkheden. Deze worden dan verscheept als één pakket. Hierdoor is er zekerheid dat de applicatie kan draaien op elke andere Linux-machine los van andere aanpassingen die gedaan zijn op die machine.

Docker is geen virtuele machine omdat het geen nieuw *operating system* installeert. De tool zorgt er wel voor dat de applicatie dezelfde Linux-kernel gebruikt als het systeem waar Docker op draait.

Docker compose

Docker compose is een opensourcetool voor het definiëren en draaien van meerdere Docker-container applicaties. Docker compose is een YAML-file waarmee applicatieservices worden geconfigureerd. Hierdoor kunnen er meerdere containers worden gestart met maar één commando.

Dockerfile

Met een Docker-file kunnen Docker *images* worden gebouwd door de instructies in de file te lezen. Een Docker-file is een tekstbestand dat alle commando's bevat die nodig zijn om een gegeven image te maken.

Docker image

Een Docker image is een file bestaand uit meerdere lagen die gebruikt wordt om code uit te voeren in een Docker container. Een image wordt gebouwd aan de hand van instructies om een uitvoerbare versie van een applicatie te bekomen. Wanneer deze image wordt gestart is er een container op basis van die image.

Nginx

Nginx is een opensourcewebserversoftware voor *reverse proxying*, *caching*, *load balancing*,…

Beats

Beats zijn lichtgewicht schippers die data verzenden van de machine waarop ze draaien naar Logstash of Elasticsearch.

Exporters

Exporters zijn tools die data verzamelen van de machine of services die op die machine draaien. Deze data wordt geconverteerd naar een speciaal formaat die Prometheus kan lezen. Hierna verstuurt de *exporter* de data niet maar zet deze ze klaar voor Prometheus via HTTP.

Solr

Solr is een opensourcezoekplatform van het Apache Lucene-project. Solr zorgt ervoor dat er een zoekmachine wordt gemaakt voor een website, databank of file.

Varnish

Varnish Cache is een opensource caching HTTP reverse proxy. Varnish wordt geïnstalleerd voor een webserver en wordt geconfigureerd om de inhoud van die server te cachen. Door het cachen hiervan zorgt Varnish dat de websiteperformantie wordt verbeterd. [2]

Memcached

Memchached is een opensource *caching* systeem gemaakt voor het versnellen van dynamische web applicaties door het verlichten van de database load. Memcached is een *in-memory key-value* opslag voor kleine brokken data van resultaten zoals database *calls*, API *calls* of pagina weergave. [3]

MySQL

MySQL is een opensourcedatabasebeheersysteem voor SQL databanken.

MSSQL

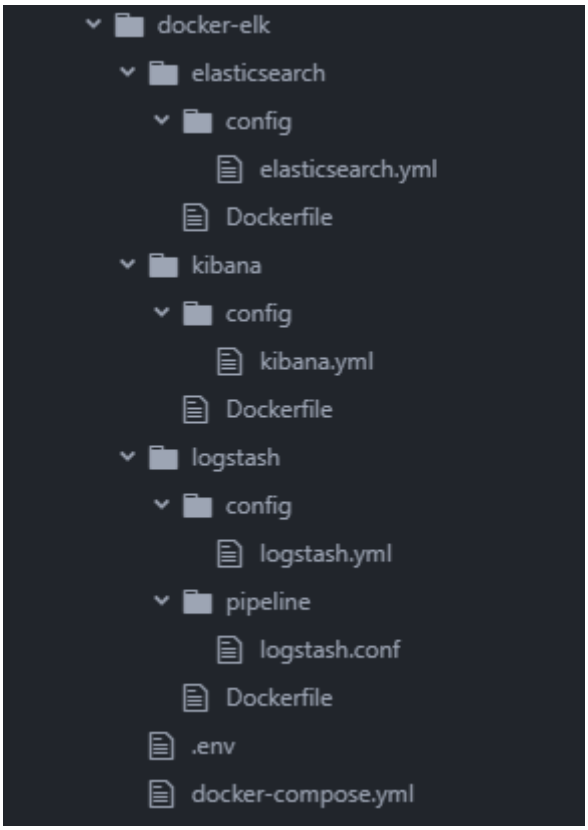
MSSQL is een databasebeersysteem voor SQL databanken gemaakt door Microsoft.

4.2 ELK

4.2.1 Mappenstructuur

Dit is de mappenstructuur van de Docker-infrastructuur. De Docker-infrastructuur is gebaseerd op die van deviantony.

In de map van elke component zit een configuratiefile en een Docker-file. In de Docker-files van elk component worden de images gebouwd. Hier kunnen er nog extra plug-ins worden geïnstalleerd. In de configuratiefile kunnen de configuraties van elk component worden aangepast.



Figuur 1 mappenstructuur docker-compose ELK

4.2.2 Elasticsearch

In de Docker-file van Elasticsearch wordt er enkel een image afgehaald van Elastic zelf. Ook is er een parameter waarmee de versie van Elasticsearch kan bepaald worden.

```
ARG ELK_VERSION
FROM docker.elastic.co/elasticsearch/elasticsearch:${ELK_VERSION}

```

Figuur 2 dockerfile ELK

In de elasticsearch.yml file wordt de Elasticsearch-instantie geconfigureerd.

Als eerste is er de variabele 'cluster.name'. Een *node* kan tot een cluster toetreden wanneer het dezelfde naam heeft als alle andere *nodes*. Ook geeft het de node een naam. Mocht deze niet zijn ingevuld, dan wordt er een random UUID gegenereerd. Na het herstarten wordt deze UUID opnieuw gegenereerd. [4] Verder is er ook nog een variabele 'network.host'. Hier moet het adres worden gegeven waarop Elasticsearch zich zal verbinden. Daarna is er nog de discovery.zen.minimum_master_nodes. Met deze variabele wordt er geconfigureerd hoeveel *nodes* er nodig zijn om een cluster te kunnen vormen. Ten slotte is er *discovery type*: deze staat op *single-node*. Hierdoor treedt deze instantie niet tot een cluster toe maar wordt die zelf master van zijn cluster.

```
cluster.name: "docker-cluster"
node.name: elastic1
network.host: 127.0.0.1
discovery.zen.minimum_master_nodes: 1
discovery.type: single-node

```

Figuur 3 elasticsearch.yml

4.2.3 Kibana

In de Docker-file van Kibana wordt er enkel een image afgehaald van Elastic zelf.

```
ARG ELK_VERSION
FROM docker.elastic.co/kibana/kibana:${ELK_VERSION}
```

Figuur 4 Docker-file Kibana

In de kibana.yml-file word de Kibana-instantie geconfigureerd.

Als eerste is er de servernaam. Dit is de naam die wordt gegeven aan de Kibana-instantie. Daarna is er de serverhost. Deze slaat terug op de backendserver. Als deze op nul wordt gezet, wordt hij onbruikbaar gemaakt. Ten slotte is er de elasticsearch.url variabele. Deze wordt meegegeven zodat Kibana weet waar het zijn *queries* kan uitvoeren.

```
server.name: kibana
server.host: "0"
elasticsearch.url: http://elasticsearch:9200

```

Figuur 5 kibana.yml-file

4.2.4 Logstash

In de Docker-file van Logstash wordt er enkel een image afgehaald van Elastic zelf.

```
ARG ELK_VERSION
FROM docker.elastic.co/logstash/logstash:${ELK_VERSION}
```

Figuur 6 Docker-file Logstash

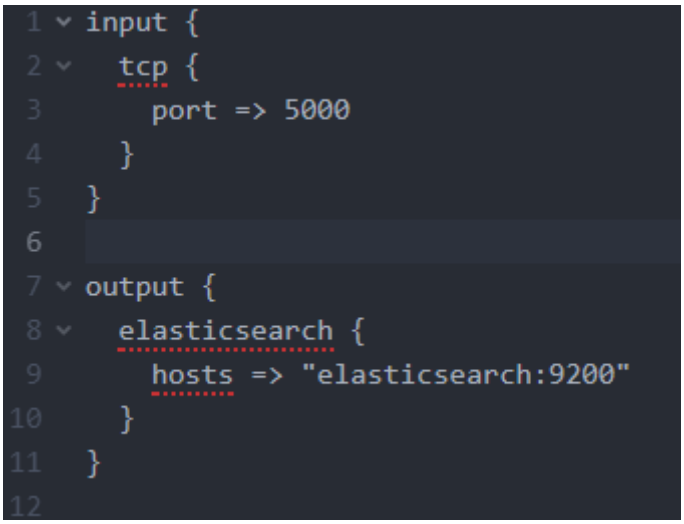
In de logstash.yml-file wordt de Logstash instantie geconfigureerd.

Voor deze file worden er twee variabelen gebruikt: http.host en path.config. De http.host is het adres voor het *metric* REST-eindpunt en de path.config variable is het pad naar de map waar al de pipelines staan.

```
http.host: "127.0.0.1"
path.config: /usr/share/logstash/pipeline
```

Figuur 7 logstash.yml

Als pipeline wordt er de standaard pipeline gebruikt. Deze stuurt alles wat hij binnenkrijgt op poort 5000 door naar Elasticsearch.

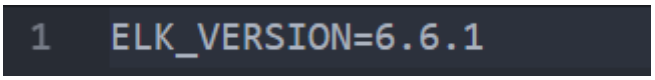


Figuur 8 pipeline Logstash

Logstash wordt in dit project niet gebruikt, maar kan voor een eventuele uitbreiding later worden gebruikt.

4.2.5 Environment file

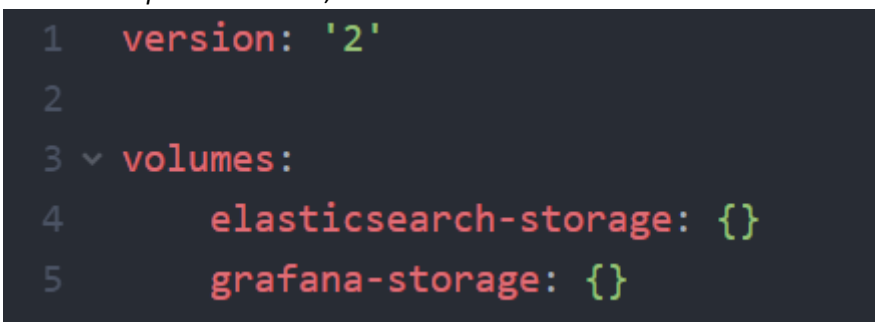
In deze file staat de versie van de ELK stack. Hierdoor kunnen de versies van alle instanties met één aanpassing worden verandert.



Figuur 9 env file ELK

4.2.6 Docker-compose

Verder wordt er gebruikgemaakt van docker-compose versie twee. Er worden twee volumes gemaakt voor *persistent* data, één voor Elasticsearch en één voor Grafana.



Figuur 10 docker-compose versie & volumes

Ook wordt er een netwerk aangemaakt met driver *bridge.* Deze maakt een privaat intern netwerk aan zodat de containers in dit netwerk met elkaar kunnen communiceren. Externe toegang wordt gegeven door poorten open te zetten naar de containers.

```
networks:

  elk:
    driver: bridge
```

Figuur 11 docker-compose netwerk

De eerste service die wordt opgezet is de Elasticsearch container. Deze wordt eerst lokaal gebouwd met als argument de versie van Elasticsearch. Verder krijgt de container de naam Elasticsearch en wordt er gebruikgemaakt van een volume en een *mount*. De *mount* dient ervoor dat er aanpassingen gedaan kunnen worden aan de configuratiefile zonder in de container te moeten inloggen. Daarna worden er nog twee poorten opengezet, poort 9200 en 9300. De poort 9200 dient als REST-eindpunt en poort 9300 is voor communicatie tussen Elasticsearch *nodes*. Als *environment* variabele wordt de *heap size* gezet op 256MB. Deze dient voor het cachen van zoekopdrachten in de RAM. Als laatste wordt de service toegevoegd aan het netwerk elk.

```
  elasticsearch:
    build:
      context: elasticsearch/
      args:
        ELK_VERSION: $ELK_VERSION
    container_name: elasticsearch
    volumes:
      - ./elasticsearch/config/elasticsearch.yml:/usr/share/elasticsearch/config/elasticsearch.yml:ro
      - elasticsearch-storage:/usr/share/elasticsearch/data
    ports:
      - "9200:9200"
      - "9300:9300"
    environment:
      ES_JAVA_OPTS: "-Xmx256m -Xms256m"
    networks:
      - elk
```

Figuur 12 docker-compose service Elasticsearch

De tweede service die wordt opgezet is de logstash container. Deze wordt eerst lokaal gebouwd met als argument de versie van logstash. Verder krijgt de container de naam logstash. Daarna worden er 2 *bind mounts* gemaakt. Deze dienen ervoor dat er makkelijk aanpassingen kunnen gedaan worden aan de configuratiefiles zonder in de container te moeten gaan. Logstash is bereikbaar via de poorten 5000 en 9600. Als *environment* variabele wordt de *heap size* gezet op 256MB en als laatste wordt de service toegevoegd aan het netwerk elk.

```
25    logstash:
26      build:
27        context: logstash/
28        args:
29          ELK_VERSION: $ELK_VERSION
30      container_name: logstash
31      volumes:
32        - ./logstash/config/logstash.yml:/usr/share/logstash/config/logstash.yml:ro
33        - ./logstash/pipeline:/usr/share/logstash/pipeline:ro
34      ports:
35        - "5000:5000"
36        - "9600:9600"
37      environment:
38        LS_JAVA_OPTS: "-Xmx256m -Xms256m"
39      networks:
40        - elk
41      depends_on:
42        - elasticsearch
```

Figuur 13 docker-compose service logstash

De derde service die wordt opgezet is de Kibana container. Deze wordt eerst lokaal gebouwd. Als argument wordt er een versie meegegeven. Verder krijgt de container de naam Kibana. Daarna wordt er een *bind mount* gemaakt. Deze dienen ervoor dat er makkelijk aanpassingen kunnen gedaan worden aan de configuratiefile zonder in de container te moeten gaan. Kibana is bereikbaar via de poort 5601. Als laatste wordt de service toegevoegd aan het netwerk elk.

```
44    kibana:
45      build:
46        context: kibana/
47        args:
48          ELK_VERSION: $ELK_VERSION
49      container_name: kibana
50      volumes:
51        - ./kibana/config/:/usr/share/kibana/config:ro
52      ports:
53        - "5601:5601"
54      networks:
55        - elk
56      depends_on:
57        - elasticsearch
```

Figuur 14 docker-compose service Kibana

Als laatste service is er de Grafana container. Bij de Grafana container wordt er gebruikgemaakt van de officiële Grafana image. De container krijgt de naam Grafana. Verder wordt er gebruik gemaakt van een volume voor *persistent* data en is Grafana bereikbaar via de poort 3000.

```
59      grafana:
60        container_name: grafana
61        image: grafana/grafana
62        volumes:
63          - grafana-storage:/var/lib/grafana
64        ports:
65          - '3000:3000'
```

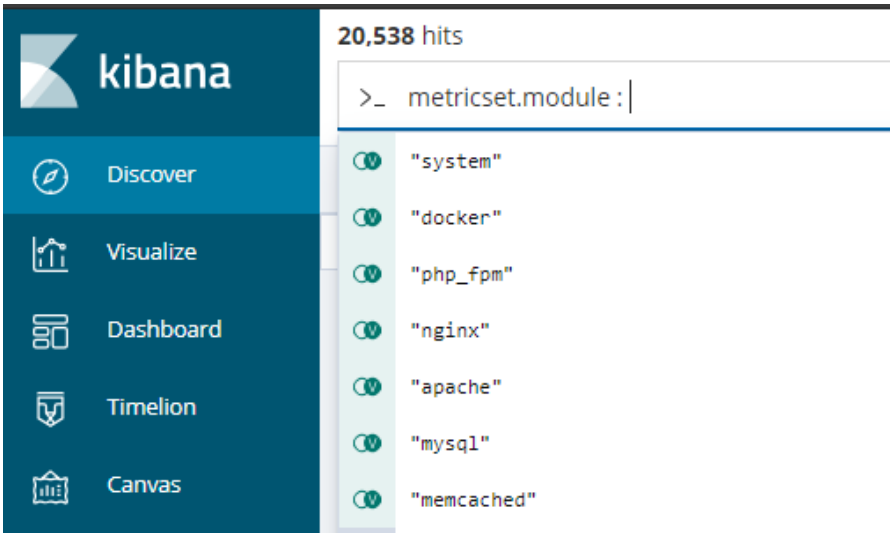
Figuur 15 docker-compose service Grafana

4.3 Nginx

De Elastic stack staat achter een Nginx met basisauthenticatie. Dit is omdat anders niet-geautoriseerde mensen de data via Kibana zouden kunnen zien of *API calls* zouden kunnen uitvoeren tegen Elasticsearch om data te verkrijgen. De Beats moeten niet achter een Nginx staan omdat Elastic stack met een push methode werkt en niets blootstelt maar de data direct stuurt naar de opgegeven instantie van Elasticsearch.

4.4 Metricbeat

Metricbeat wordt geïnstalleerd op de targetserver. Deze haalt periodiek *metrics* op van de server en verscheept die dan naar de Elasticsearch server. Verder worden op Metricbeat verschillende modules geactiveerd om de volgende services te monitoren: Apache, MySQL, Nginx, Memcached, PHP_FPM, Docker en systeem *metrics*.



Figuur 16 ELK stack modules

4.4.1 Apache module

```
- module: apache
  metricsets: ["status"]
  period: 10s
  enabled: true
  hosts: ["http://x.x.x.x:xxxx"]
```

Figuur 17 ELK stack apache module config

Deze module haalt de gegevens van Apache op door te kijken naar de Apache mod_status module. Metricbeat *scrapet* de serverstatus data van de webpagina die wordt gegenereerd door mod_status.

Hoe de serverstatuspagina wordt geactiveerd staat beschreven in Apache *exporter*.

4.4.2 Mysql module

```
- module: mysql
  metricsets:
    - status
  period: 10s
  hosts: ["@unix(/var/run/mysqld/mysqld.sock)/"]
  username: root
  password: ${MySQL_PWD}
```

Figuur 18 ELK stack Mysql module config

Deze module haalt de gegevens van MySQL door een "SHOW GLOBAL STATUS;" SQL *query* uit te voeren. De module connecteert met de MySQL instantie door middel van een unix socket. Verder wordt het paswoord van de MySQL user meegegeven met een *keystore*. Hierdoor wordt het paswoord nergens *plain text* opgeslaan.

4.4.3 Nginx module

```
- module: nginx
  metricsets: ["stubstatus"]
  period: 10s
  enabled: true
  hosts: ["127.0.0.1:8080"]
  server_status_path: "basic_status"
```

Figuur 19 ELK stack Nginx module config

Deze module haalt de gegevens van de Nginx ngx_http_stub_status module. Metricbeat *scrapet* de serverstatus data van de webpagina die wordt gegeneerd door ngx_http_stub_status.

Hoe de stub_status pagina wordt geactiveerd staat beschreven in Nginx *exporter*.

4.4.4 Docker module

```
- module: docker
  metricsets:
    - container
    - cpu
    - diskio
    - event
    - healthcheck
    - info
    - memory
    - network
  period: 10s
  hosts: ["unix:///var/run/docker.sock"]
```

Figuur 20 ELK stack Docker module config

Deze module haalt de gegevens van Docker containers. Metricbeat connecteert via de Docker-socket met de Docker-daemon. Dit is het ingangspunt voor de Docker API.

4.4.5 PHP_FPM module

```
- module: php_fpm
  metricsets:
    - pool
    - process
  period: 10s
  enabled: true
  hosts: ["localhost"]
  status_path: "/status"
```

Figuur 21 ELK stack PHP-FPM module config

Deze module haalt de gegevens van de service PHP-FPM. Hiermee kunnen er *metrics* van verschillende PHP-pools en PHP-processen worden verzameld. Verder wordt het pad en de host meegegeven.

Om deze gegevens te verkrijgen moet de PHP-statuspagina worden geactiveerd. Dit staat beschreven in PHP-FPM *exporter*.

4.4.6 Systeem module

```
- module: system
  period: 2s
  metricsets:
    - cpu
    - load
    - memory
    - network
    - process
    - process_summary
    - core
    - diskio
    - socket
  processes: ['.*']
  process.include_top_n:
    by_cpu: 15
    by_memory: 15
  cpu.metrics: ["percentages", "normalized_percentages"]
  core.metrics: ["percentages"]


- module: system
  period: 30s
  metricsets:
    - filesystem
    - fsstat
  processors:
  - drop_event.when.regexp:
      system.filesystem.mount_point: '^/(sys|cgroup|proc|dev|etc|host|lib)($|/)'

- module: system
  period: 15m
  metricsets:
```

Figuur 22 ELK stack systeem module config

Deze module verzamelt de systeem *metrics* van de server. Deze dienen om een algemeen beeld te krijgen van de belasting op de server. Ook wordt er ook gegevens over processen verzameld om een beter beeld welk processen de server het meest belasten.

4.5 Heartbeat

Heartbeat wordt geïnstalleerd op de Elastic stack server zelf. Van hieruit worden er URL's toegevoegd waarop Heartbeat moet controleren of deze bereikbaar zijn. Heartbeat controleert niet enkel of het bereikbaar is, maar ook wordt er een controle gedaan op de inhoud van de body die het krijgt terug-gestuurd.

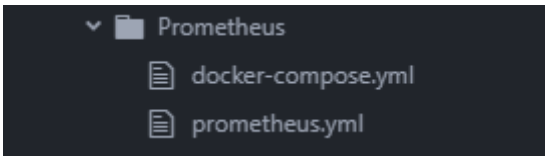
```
heartbeat.monitors:
- type: http

  # List or urls to query
  urls: ["http://gui.cp4dev.be/login"]

  # Configure task schedule
  schedule: '@every 10s'
  check.request:
    method: GET

  check.response:
    status: 200
    body: "Forgotten password?"

output.elasticsearch:
  hosts: ["185.142.227.115:8081"]
  protocol: "http"
  username: "admin"
  password: ${nginx}
```

Figuur 23 ELK stack heartbeat.yml

4.6 Prometheus

4.6.1 mappenstructuur

Dit is de mappenstructuur van de Docker-infrastructuur. Er is een docker-compose file waarmee de Docker-infrastructuur wordt opgezet en de prometheus.yml file waarmee de Prometheus server wordt geconfigureerd.



Figuur 24 mappenstructuur Prometheus infrastructuur

De prometheus.yml file is de configuratiefile van de Prometheus-server. Hier worden alle *scrape* targets en het interval van de *scrape* gedefinieerd. Ook kan het *scrape* interval per job gedefinieerd worden. Verder wordt hier ook verwezen naar de alertmanager en worden de *alert rules* ingeladen.

```yaml
global:

  scrape_interval: 30s

rule_files:
  - alert.rules.yml

alerting:
  alertmanagers:
  - static_configs:
    - targets:
      - localhost:9093
```

Figuur 25 prometheus.yml alerts

```
scrape_configs:
  - job_name: 'localhost'

    scrape_interval: 30s
    basic_auth:
      username: 'admin'
      password_file: '/etc/prometheus/.htpasswd'

    static_configs:
      - targets: ['x.x.x.x:8083']
        labels:
          group: 'remote php-fpm'

      - targets: ['x.x.x.x:8082']
        labels:
          group: 'docker metrics'

      - targets: ['x.x.x.x:8085']
        labels:
          group: 'docker mssql'

      - targets: ['x.x.x.x:8088']
        labels:
          group: 'memcached'

      - targets: ['x.x.x.x:8090']
        labels:
          group: 'varnish'

      - targets: ['x.x.x.x:8093']
        labels:
          group: 'haproxy'

      - targets: ['x.x.x.x:8094']
        labels:
          group: 'metrics'
```

Figuur 26 prometheus.yml localhost

```
  - job_name: 'localhost slow request'
    scrape_interval: 60s

    basic_auth:
      username: 'admin'
      password_file: '/etc/prometheus/.htpasswd'

    static_configs:
      - targets: ['x.x.x.x:9854']
        labels:
          group: 'docker solr'
```

Figuur 27 prometheus.yml localhost slow request

```yaml
- job_name: 'ServerA'
  basic_auth:
    username: 'admin'
    password_file: '/etc/prometheus/.htpasswd'

  static_configs:

    - targets: ['x.x.x.x:8086']
      labels:
        group: 'metrics'

    - targets: ['x.x.x.x:8084']
      labels:
        group: 'apache'

    - targets: ['x.x.x.x:8083']
      labels:
        group: 'nginx'

    - targets: ['x.x.x.x:8082']
      labels:
        group: 'mysql'

    - targets: ['x.x.x.x:8085']
      labels:
        group: 'php-fpm-exporter'

    - targets: ['x.x.x.x:8087']
      labels:
        group: 'processes'
```

Figuur 28 prometheus.yml ServerA

```yaml
- job_name: 'ServerB'
  scrape_interval: 40s
  basic_auth:
    username: 'admin'
    password_file: '/etc/prometheus/.htpasswd'
  static_configs:
    - targets: ['x.x.x.x:8081']
      labels:
        group: 'wmi_exporter'

    - targets: ['x.x.x.x:8084']
      labels:
        group: 'MSSQL_exporter'
```

Figuur 29 prometheus.yml ServerB

```
- job_name: 'ServerC'
  basic_auth:
    username: 'admin'
    password_file: '/etc/prometheus/.htpasswd'
  static_configs:
    - targets: ['x.x.x.x:8081']
      labels:
        group: 'process'
```

Figuur 30 prometheus.yml ServerC

```
- job_name: 'blackbox'
  metrics_path: /probe
  params:
    module: [http_2xx]
  basic_auth:
    username: 'admin'
    password_file: '/etc/prometheus/.htpasswd'
  static_configs:
    - targets:
      - 'http://website.be/login'
      - 'http://website2.be/#home'
      - 'http://x.x.x.x:8083'
  relabel_configs:
    - source_labels: [__address__]
      target_label: __param_target
    - source_labels: [__param_target]
      target_label: instance
    - target_label: __address__
      replacement: x.x.x.x:8084
```

Figuur 31 prometheus.yml blackbox

4.6.2 Docker-compose

Voor de docker-compose-file wordt er gebruik gemaakt van docker-compose versie 3.1. Er worden 2 volumes gemaakt voor *persistent* data, 1 voor Prometheus en 1 voor Grafana. Hierdoor wordt de data van de Grafana en Prometheus server behouden na het afsluiten van de Docker-container. Verder wordt er nog een bestaand netwerk gebruikt.

```
version: "3.1"

volumes:
    prometheus_data: {}
    grafana-storage: {}
networks:
    prometheus:
        external :
            name: prometheus
```

Figuur 32 prometheus docker-compose deel1

De eerste service die wordt opgezet, is de Prometheus-server. Voor deze service wordt er gebruik gemaakt van de officiële Prometheus-image. De configuratiefiles en *credentials* worden doorgegeven via een *mount*. De *credentials* worden doorgestuurd voor authenticatie met Nginx wanneer de gegevens van de *exporters* worden verzameld . Verder wordt voor de overige data nog gebruik gemaakt van een Docker-volume.

Vervolgens worden er nog verschillende commando's uitgevoerd zoals het opstarten van Prometheus met de gespecifieerde configuratiefile en het pad waar de Prometheus databank staat. Daarna is er nog een variabele waarin kan worden meegegeven hoelang de data wordt opgeslagen en wordt er verwezen naar de console templates die gebruikt zullen worden.

Omdat Prometheus achter een Nginx-proxyserver met basisauthenticatie staat, luistert deze enkel naar localhost op poort 9090. Ten slotte wordt Prometheus toegevoegd aan het Prometheus netwerk.

```
prometheus:
  image: prom/prometheus
  volumes:
    - ./configs/:/etc/prometheus/
    - prometheus_data:/prometheus
  command:
    - '--config.file=/etc/prometheus/prometheus.yml'
    - '--storage.tsdb.path=/prometheus'
    - '--storage.tsdb.retention=31d'
    - '--web.console.libraries=/usr/share/prometheus/console_libraries'
    - '--web.console.templates=/usr/share/prometheus/consoles'
  ports:
    - 127.0.0.1:9090:9090
  networks:
    - prometheus
```

Figuur 33 prometheus docker-compose deel2

De laatste service die wordt opgezet is Grafana. Voor deze service wordt gebruik gemaakt van de officiële Grafana image. Verder is er een *mount* om de Grafana data bij te houden. Omdat er al authenticatie is ingebouwd in Grafana luistert deze op poort 3000 naar alle IP-adressen. Ten slotte wordt Grafana gekoppeld aan het Prometheus netwerk.

```
28    grafana:
29      image: grafana/grafana
30      volumes:
31        - grafana-storage:/var/lib/grafana
32      ports:
33        - '3000:3000'
34      networks:
35        - prometheus
```

Figuur 34 prometheus docker-compose deel3

4.6.3 Nginx

Op de Prometheus server en de targetservers waarop de *exporters* staan wordt er gebruik gemaakt van basisauthenticatie. Dit is omdat Prometheus geen authenticatie heeft en omdat er geen firewall voor de Prometheus server staat.

4.7 Exporters

4.7.1 Nginx

Op de targetservers waar de *exporters* draaien, wordt er gebruik gemaakt van basisauthenticatie. Dit is omdat anders andere Prometheus-servers ook van deze targets *metrics* zouden kunnen afhalen. Ook staat Nginx standaard op de webservers. In deze voorbeelden zijn de URL, IP-adressen en poorten veranderd wegens securitymaatregelen.

4.7.2 Node exporter

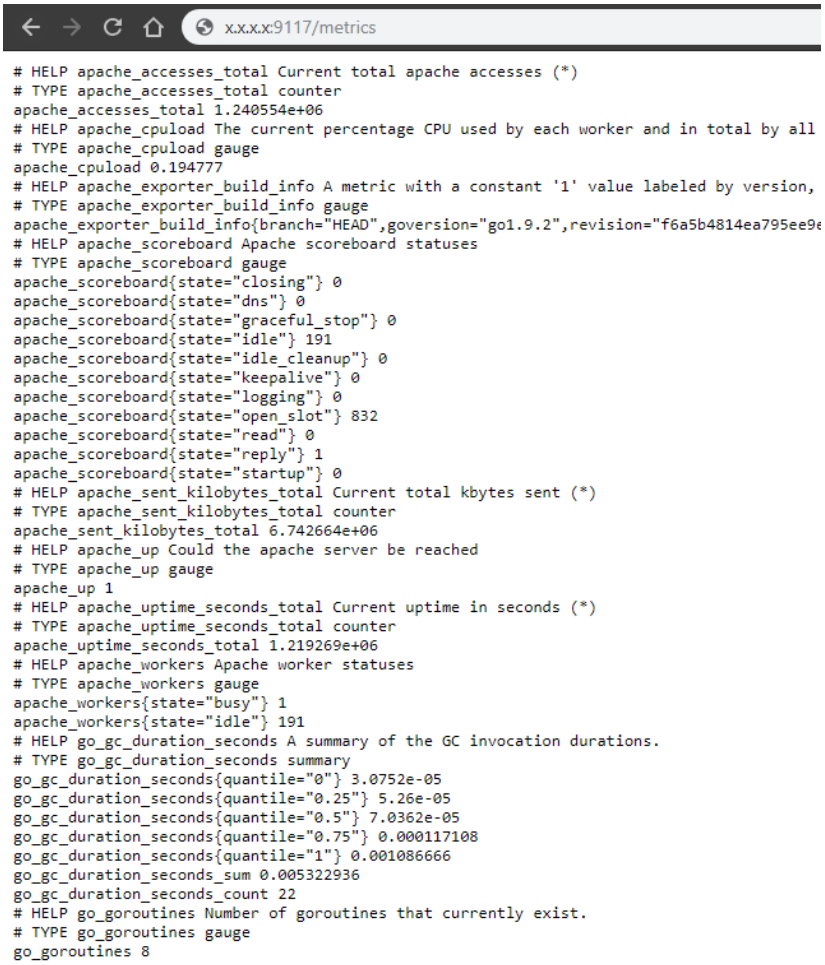
De *node exporter* dient voor het ophalen van de hardware en OS *metrics* en heeft veel verschillende collectors zoals CPU, diskstats, filesystem,… Dit zijn de default *collectors* waarmee de basissysteem *metrics* mee worden gemonitord. Als extra collectors wordt NTP nog gemonitord voor eventuele *time drift* tussen de servers. Ten slotte wordt Supervisord nog gemonitord om de *uptime* van services te monitoren.

Figuur 35 metrics pagina node exporter

Deze *metrics* worden dan blootgesteld op http://localhost:9100/metrics en van hieruit worden de *metrics* verzameld door Prometheus.

4.7.3 Apache exporter

De Apache *exporter* monitort de Apache-service door gebruik te maken van de mod_status statistieken. Hierop staat de *uptime* van Apache, CPU load van Apache, aantal connecties,… Deze *metrics* worden dan blootgesteld op http://localhost:9117/metrics. Hier worden dan de *metrics gescraped* door Prometheus.



Figuur 36 metrics pagina Apache exporter

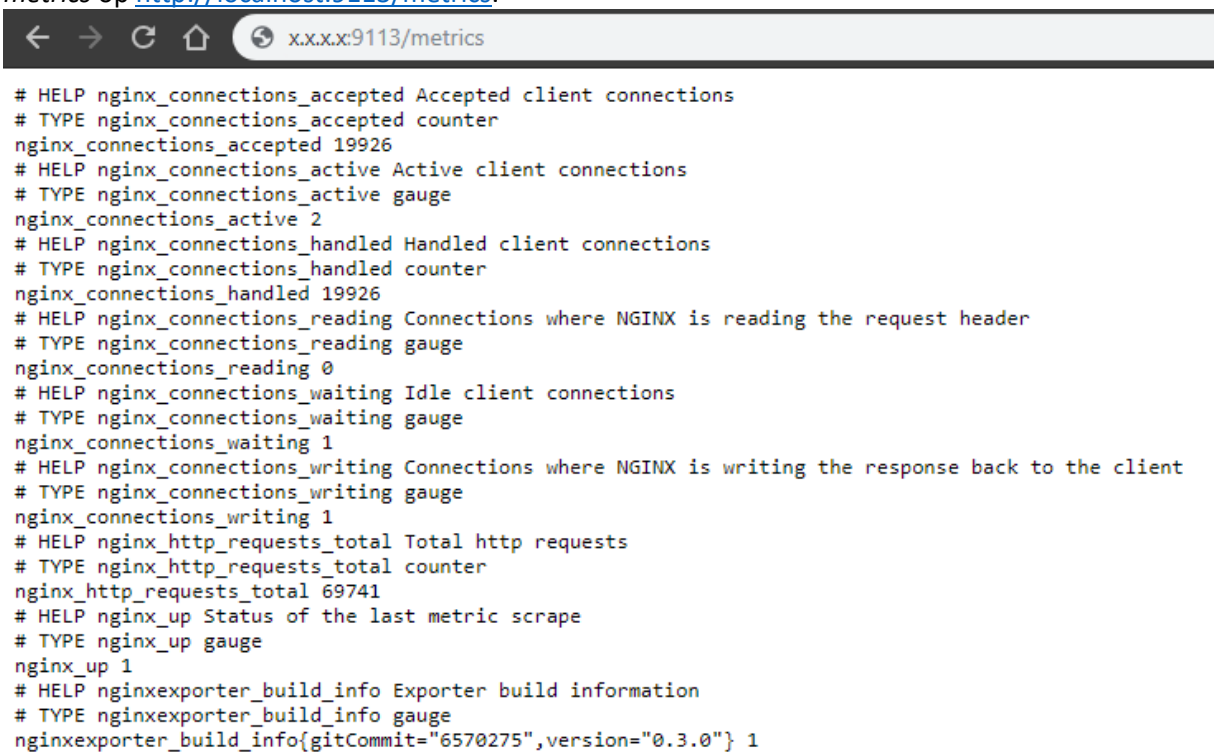
Om te zorgen dat de *exporter* de gegevens van Apache kan lezen, moet de serverstatuspagina worden aangezet. Ook wordt er ingesteld dat deze website enkel beschikbaar is voor 127.0.0.1 zodat enkel deze server hier aan kan.

```
1. <Location /server-status>
2.     SetHandler server-status
3.     Require local
4.     Require ip 127.0.0.1 ::1
5. </Location>
6.
7. ExtendedStatus On
```

Hierna kan de *exporter* de data lezen en transformeert deze gegevens in een *format* die Prometheus kan lezen op http://localhost:9117/metrics.

4.7.4 Nginx exporter

De Nginx *exporter* monitort de Nginx-service door gebruik te maken van de stub_status pagina. Op deze pagina staan de *metrics* van de Nginx-service. Deze worden geconverteerd naar een *format* die Prometheus kan lezen waarna ze blootgesteld worden via HTTP. Daarna verzamelt Prometheus deze *metrics* op http://localhost:9113/metrics.



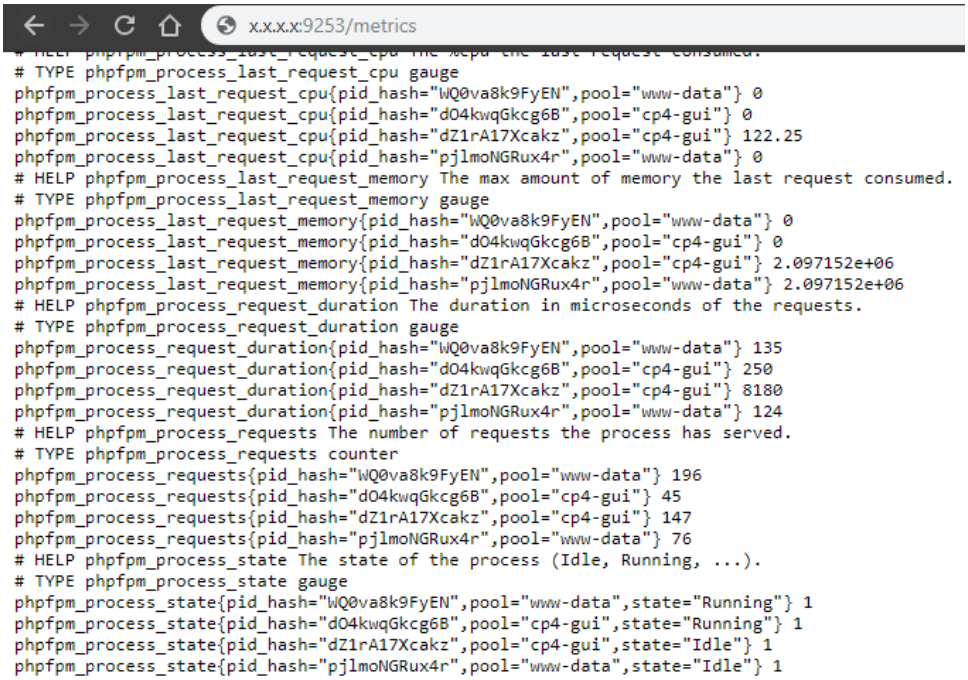
Figuur 37 metrics pagina nginx exporter

Om te zorgen dat de *exporter* de gegevens van Nginx kan lezen, moet de stub_status pagina worden aangezet. Door enkel 127.0.0.1 toe te laten is deze website enkel toegankelijk voor de server zelf.

```
1. server {
2.     listen *:8080;
3.     location /basic_status {
4.       stub_status on;
5.       access_log off;
6.       allow 127.0.0.1;
7.       deny all;
8.     }
9. }
```

4.7.5 PHP-FPM exporter

De PHP-FPM *exporter* monitort één of meerdere PHP-FPM *pools* door gebruik te maken van de statuspagina van PHP. De *exporter* connecteert via het FastCGI adres naar de statuspagina. Daar worden de *metrics* omgezet naar een andere *format* die Prometheus kan lezen en worden dan blootgesteld op http://localhost:9253/metrics. Hier worden de *metrics* dan verzameld door Prometheus.



Figuur 38 metrics pagina PHP-FPM exporter

Om de statuspagina online te krijgen moet deze eerst geactiveerd worden in de configuratiefile van de *pool*. Dit gebeurt door de volgende regel.

```
pm.status_path = /status
```

Figuur 39 PHP pool status pagina config

4.7.6 MSSQL exporter

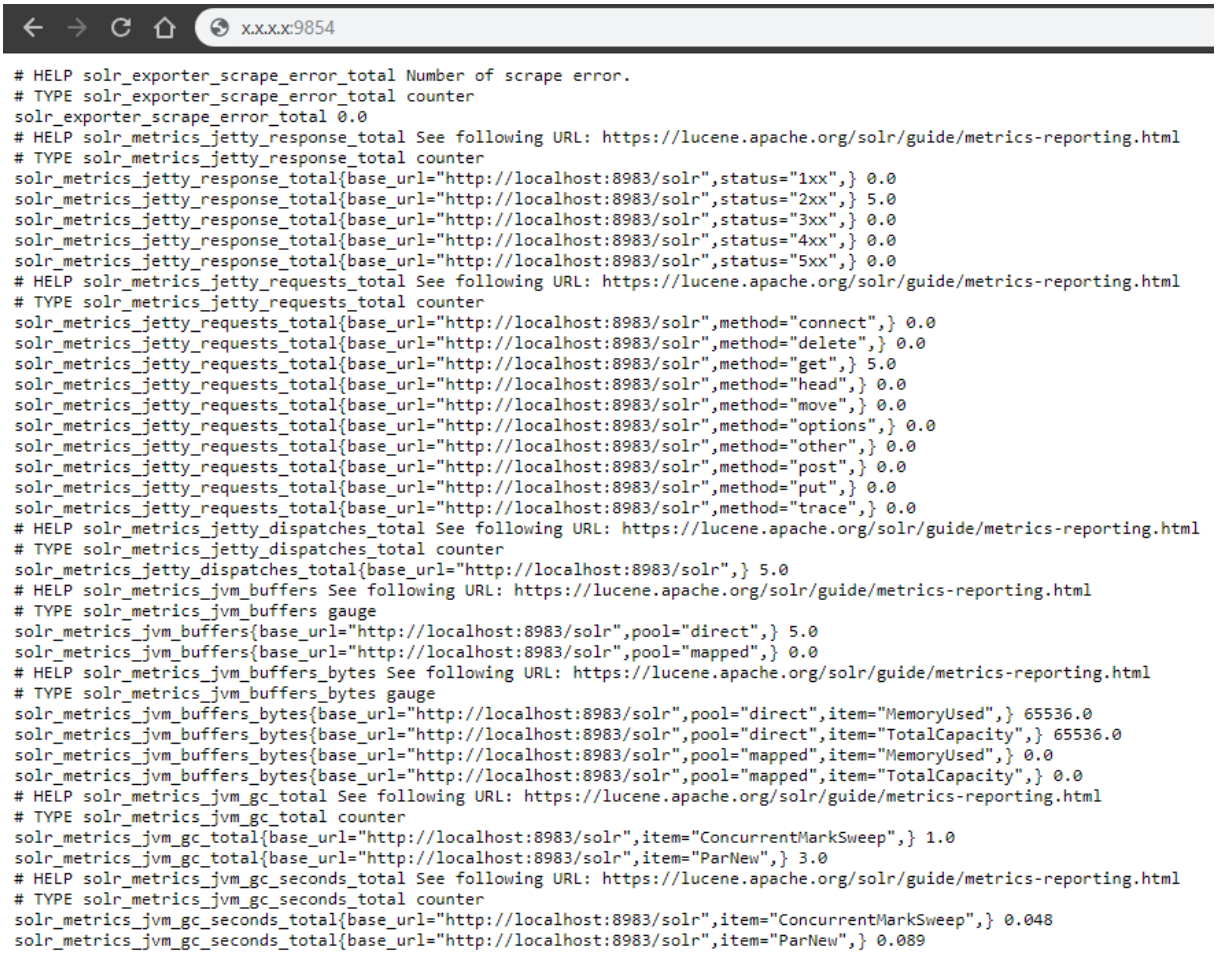
De MSSQL *exporter* monitort een MSSQL-databank. De *exporter* wordt opgezet via een Docker-container en verzamelt *metrics* zoals het aantal actieve connecties, aantal user errors de seconde, aantal transact-SQL commandos ontvangen per seconde,… De metrics worden dan omgezet naar een andere *format* zodat Prometheus deze kan lezen en worden dan blootgesteld op http://localhost:4000/metrics. Hier worden de *metrics* verzameld door Prometheus.


```
# HELP up UP Status
# TYPE up gauge
up 1

# HELP mssql_instance_local_time Number of seconds since epoch on local instance
# TYPE mssql_instance_local_time gauge
mssql_instance_local_time 1553781311

# HELP mssql_connections Number of active connections
# TYPE mssql_connections gauge
mssql_connections{database="master",state="current"} 44
mssql_connections{database="null",state="current"} 1

# HELP mssql_deadlocks Number of lock requests per second that resulted in a deadlock since last restart
# TYPE mssql_deadlocks gauge
mssql_deadlocks 0

# HELP mssql_user_errors Number of user errors/sec since last restart
# TYPE mssql_user_errors gauge
mssql_user_errors 8549

# HELP mssql_kill_connection_errors Number of kill connection errors/sec since last restart
# TYPE mssql_kill_connection_errors gauge
mssql_kill_connection_errors 0

# HELP mssql_log_growths Total number of times the transaction log for the database has been expanded last restart
# TYPE mssql_log_growths gauge
mssql_log_growths{database="tempdb"} 0
mssql_log_growths{database="msdb"} 0
mssql_log_growths{database="model"} 0
mssql_log_growths{database="mssqlsystemresource"} 0
mssql_log_growths{database="master"} 6
```

Figuur 40 metrics pagina MSSQL exporter

4.7.7 Solr exporter

Solr *exporter* monitort Solr voor *metrics* die komen van de *metric* API, facet *counts* van zoekopdrachten en responses, Collections API *commands* en PingRequestHandler verzoeken. De Solr *exporter* zit ingebouwd in Solr zelf. Verder worden de *metrics* omgezet naar een *format* die Prometheus kan lezen en worden ze blootgesteld op [http://localhost:9854/metrics.](http://localhost:9854/metrics)



Figuur 41 metrics pagina Solr exporter

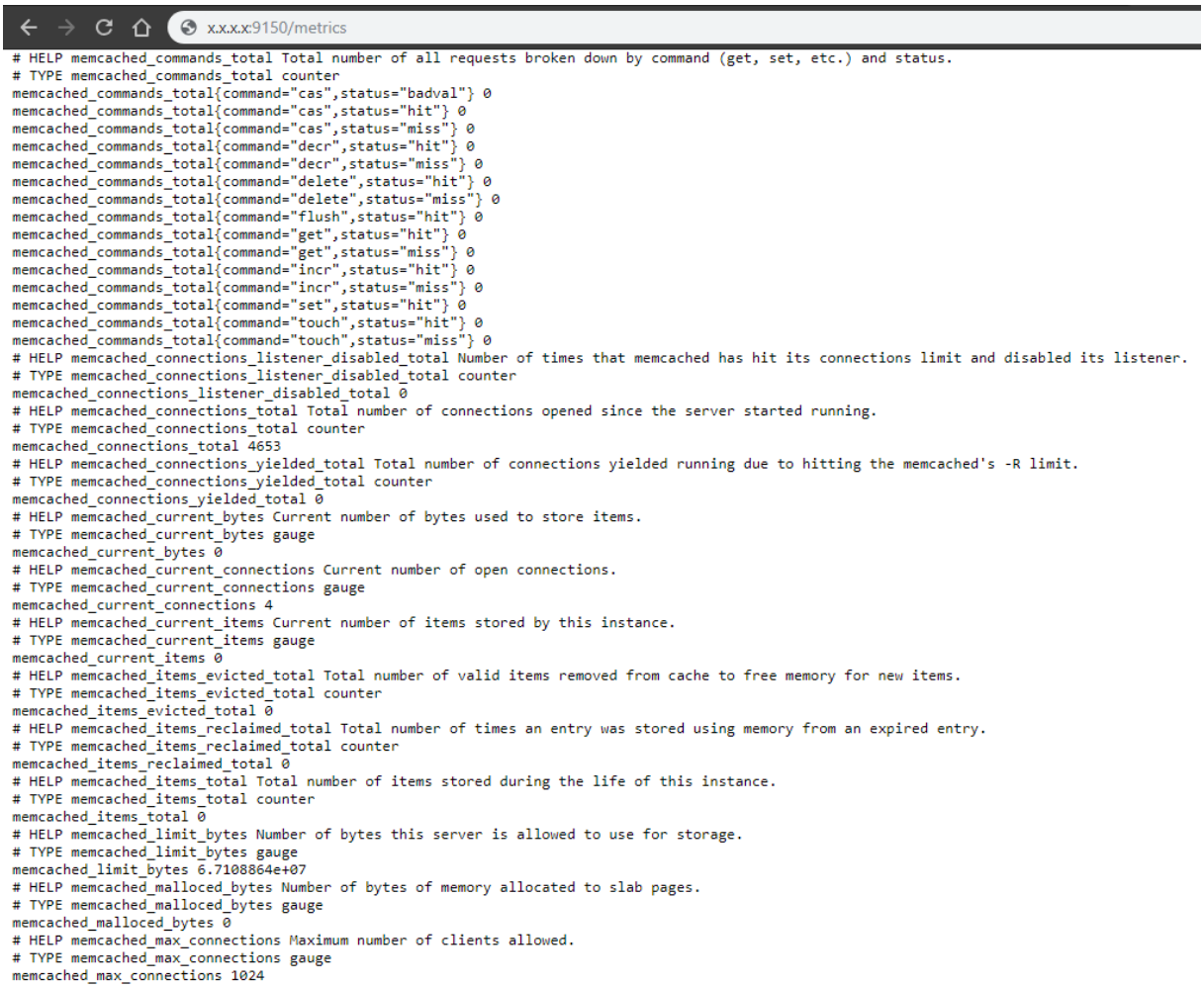
4.7.8 Container exporter

De container *exporter* vraagt een lijst van containers die aan het draaien zijn op de host door te communiceren met cAdvisor. Daarna verzamelt het de *metrics* door gebruik te maken van libcontainer en DockerClient. Verder worden de *metrics* geconverteerd naar een format die Prometheus kan lezen en worden ze blootgesteld op http://localhost:8080/metrics.

Figuur 42 metrics pagina container exporter

4.7.9 Memcached exporter

De Memcached *exporter* verzamelt *metrics* van een Memcached-server, converteert deze *metrics* naar een *format* zodat Prometheus deze kan lezen en stelt ze dan bloot op http://localhost:9150/metrics.



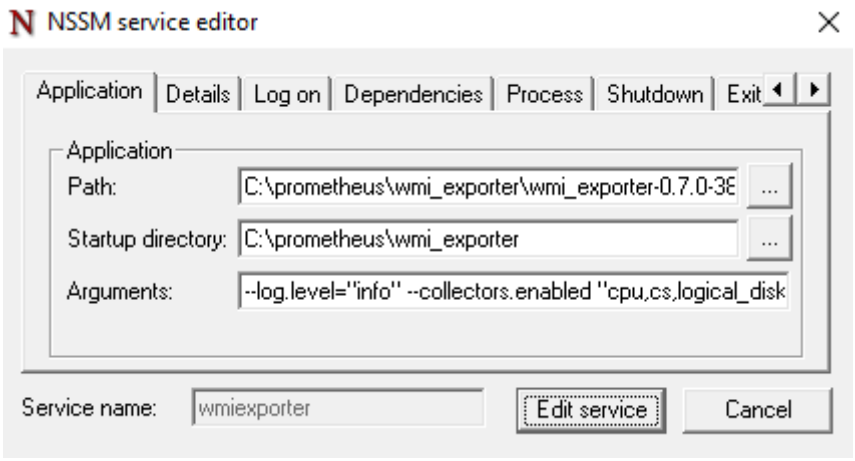
Figuur 43 metrics pagina memcached exporter

4.7.10 WMI exporter

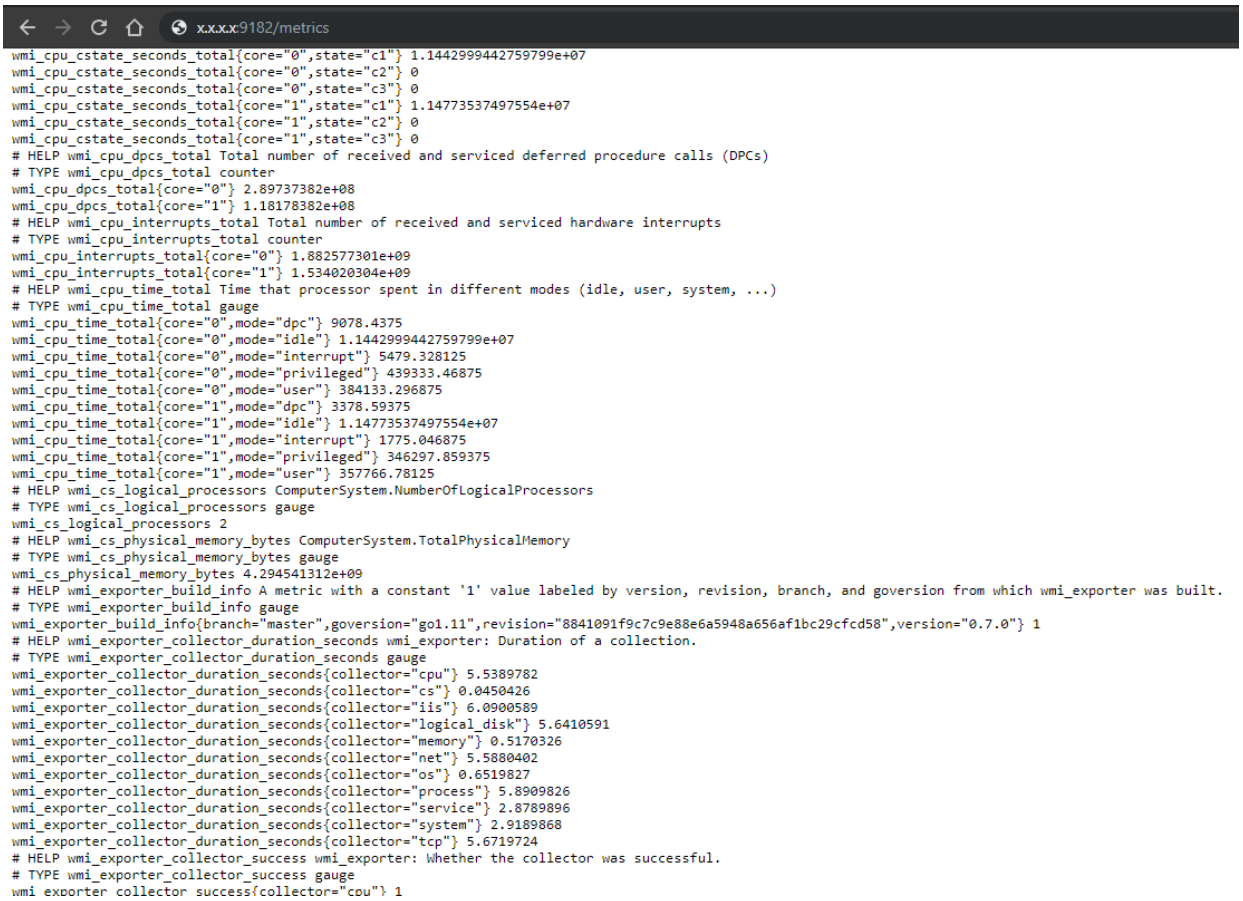
De WMI *exporter* wordt gebruikt voor het monitoren van een Windows-machine alsook voor de componenten die extra worden geïnstalleerd op de server. In dit voorbeeld wordt de *exporter* gebruikt voor het verzamelen van de basisperformantiegegevens alsook de processen en IIS. Verder wordt het HTTP eindpunt nog toegevoegd. De *exporter* wordt ingesteld zodat deze enkel luistert naar connecties van de machine zelf. Dit is omdat Nginx nog voor de *exporter* staat met basisauthenticatie.

De *exporter metrics* worden standaard blootgesteld op pad "/metrics. Verder wordt er van deze .exe file een Windows service gemaakt. Dit gebeurt via NSSM.

```
1. c:\prometheus\wmi_exporter\wmi_exporter-0.7.0-386.exe --
   log.level="info" --collectors.enabled "cpu,cs,logi-
   cal_disk,net,os,service,system,tcp,process,iis,memory" --teleme-
   try.addr="127.0.0.1:9182"
```



Figuur 44 NSSM WMI exporter service



Figuur 45 metrics pagina WMI exporter

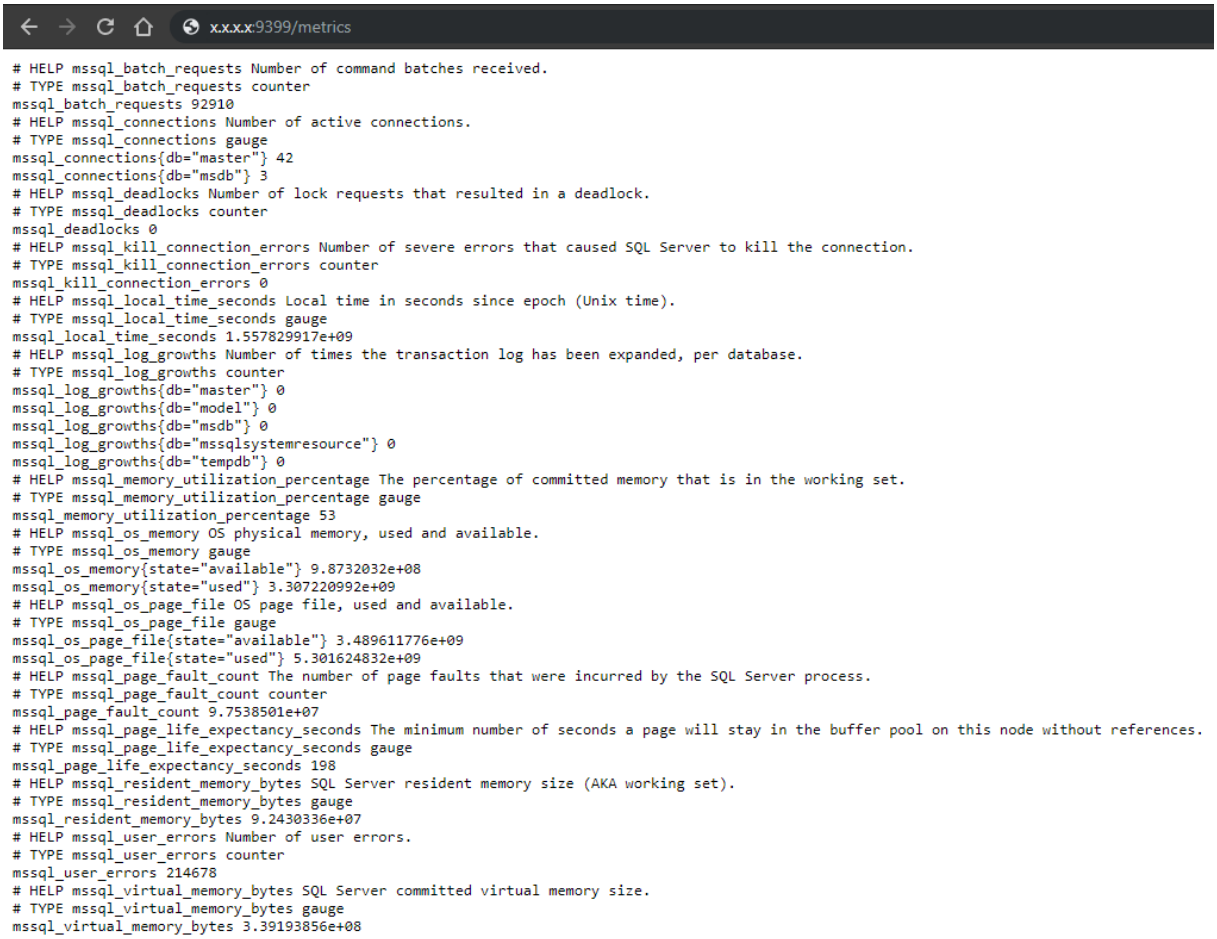
De WMI *exporter* wordt op een langere interval gezet dan andere *exporters* door het grote aantal *metrics* dat wordt verzameld.

```
- job_name: 'windows'
  scrape_interval: 40s
  basic_auth:
    username: 'admin'
    password_file: '/etc/prometheus/.htpasswd'

  static_configs:
    - targets: ['x.x.x.x:8081']
      labels:
        group: 'wmi_exporter'
```

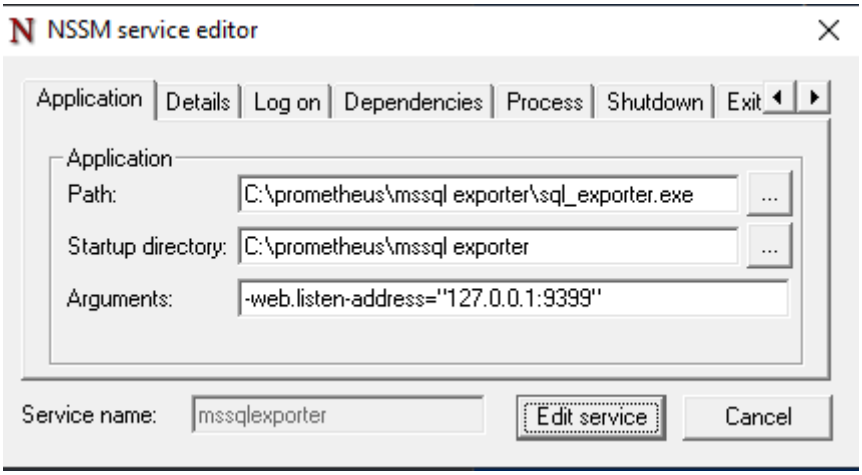
Figuur 46 config prometheus WMI exporter

4.7.11 MSSQL exporter Windows

De MSSQL *exporter* wordt gebruikt voor het monitoren van een SQL-server op een Windows-machine. De *exporter* verzamelt gegevens zoals actieve connecties, *deadlocks*, gestopte connecties, geheugengebruik van de database,… Deze worden dan geconverteerd naar het Prometheus *format* en blootgesteld op http://localhost:9399/metrics. Hier worden de *metrics* verzameld door Prometheus.



Figuur 47 metrics pagina MSSQL exporter



Figuur 48 NSSM service MSSQL exporter

```yaml
global:

  scrape_timeout_offset: 500ms

  min_interval: 0s

  max_connections: 3

  max_idle_connections: 3

target:
  data_source_name: 'sqlserver://127.0.0.1:1433'
  collectors: [mssql_standard]
collector_files:
  - "*.collector.yml"
```

Figuur 49 SQL_export.yml

4.7.12 Blackbox exporter

Met de blackbox *exporter* kunnen eindpunten zoals websites over HTTP,HTTPS, DNS,TCP en ICMP worden gepeild of ze nog bereikbaar zijn. In deze situatie wordt er enkel de HTTP-module gebruikt. Dit is omdat er enkel wordt gemonitord op de bereikbaarheid van de websites en voor de controle op het certificaat.

```yaml
modules:
  http_2xx:
    prober: http
    timeout: 5s
    http:
      valid_http_versions: ["HTTP/1.1", "HTTP/2"]
      valid_status_codes: []  # Defaults to 2xx
      method: GET

      fail_if_body_matches_regexp:
        - "502 Bad Gateway"
      fail_if_ssl: false
```

Figuur 50 blackbox.yml

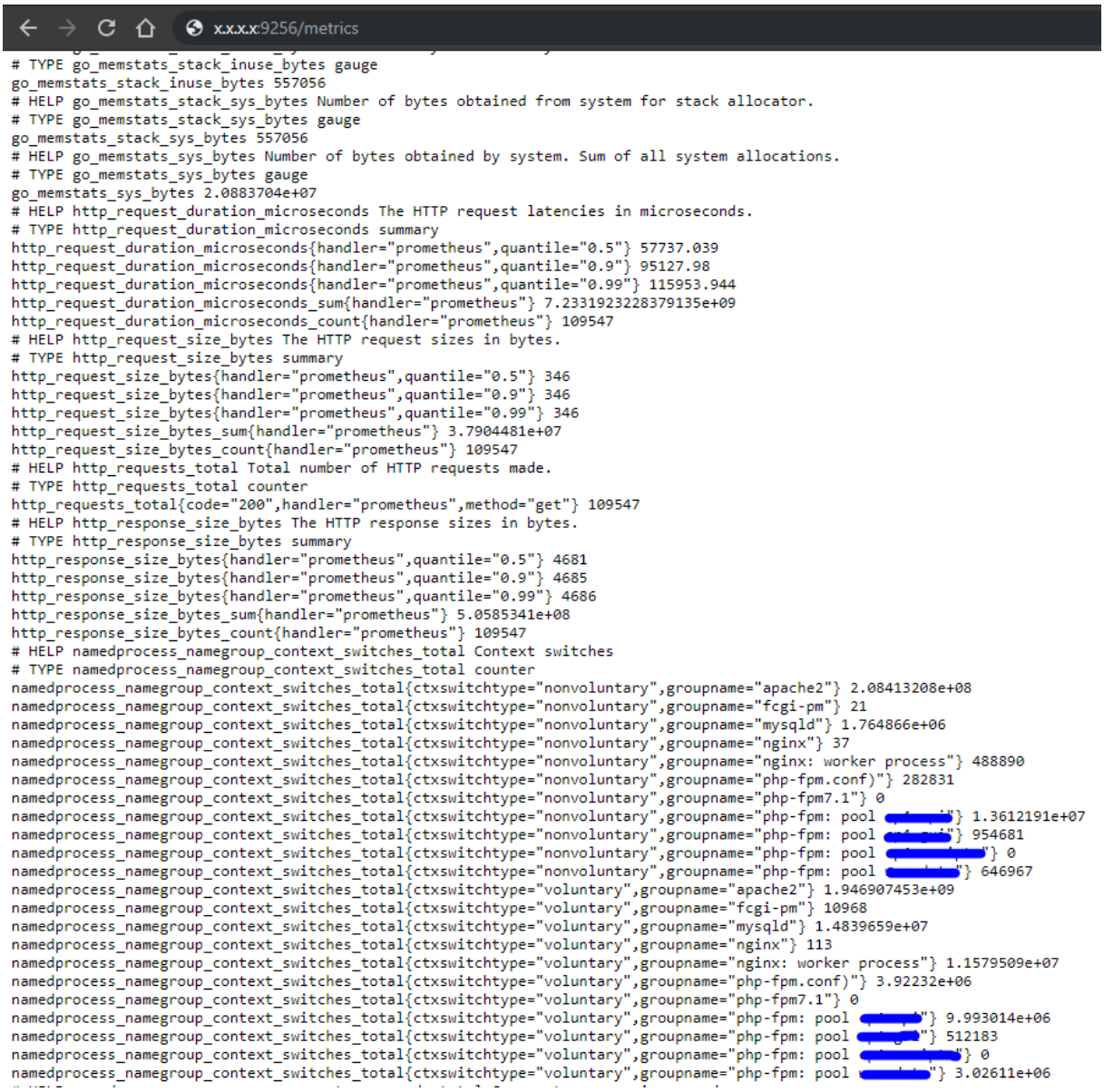
Verder worden de targets op wie de checks gebeuren gedefinieerd in de Prometheus configuratie.

```yaml
- job_name: 'blackbox'
  metrics_path: /probe
  params:
    module: [http_2xx]
  basic_auth:
    username: 'admin'
    password_file: '/etc/prometheus/.htpasswd'
  static_configs:
    - targets:
      - 'https://foobar.com'
      - 'https://foobar2.com'
      - 'https://foobar3.com'
  relabel_configs:
    - source_labels: [__address__]
      target_label: __param_target
    - source_labels: [__param_target]
      target_label: instance
    - target_label: __address__
      replacement: x.x.x.x:8084
```

Figuur 51 blackbox exporter prometheus.yml

4.7.13 Process exporter

De *process exporter* zorgt ervoor dat de actieve processen op een machine worden verzameld. Verder worden deze gegevens geconverteerd naar een formaat dat Prometheus kan lezen. Daarna worden deze blootgesteld op http://localhost:9256 waar ze worden opgehaald door Prometheus.



Figuur 52 metrics pagina process exporter

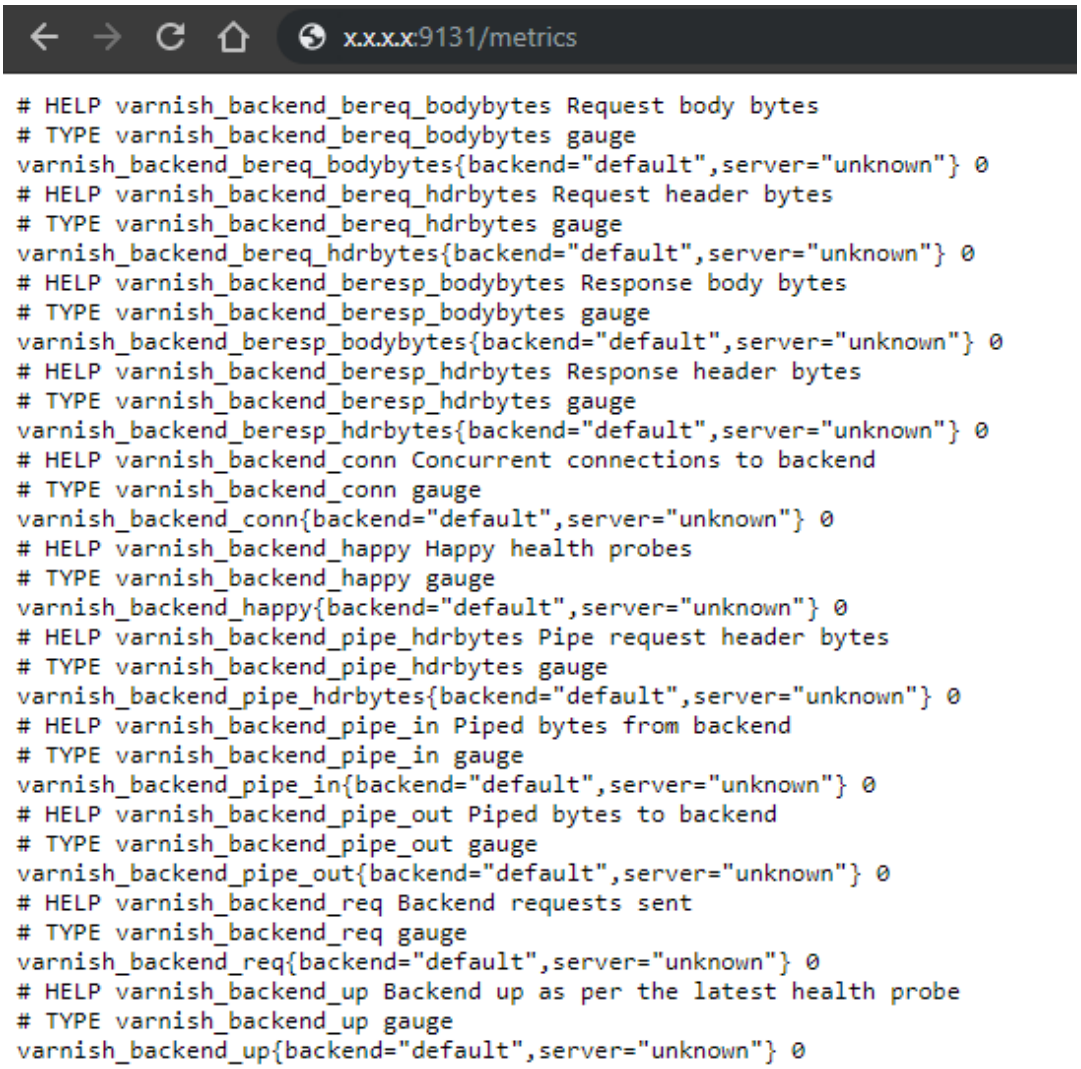
Met de config.yml worden de processen gedefinieerd die moeten worden gemonitord. De process *exporter* zal niet alle processen monitoren omdat dit anders een grote impact op de server kan hebben.

```
process_names:
  - comm:
    - apache2
    - nginx
    - php-fpm7.1
    - mysqld
```

Figuur 53 process exporter config.yml

4.7.14 Varnish exporter

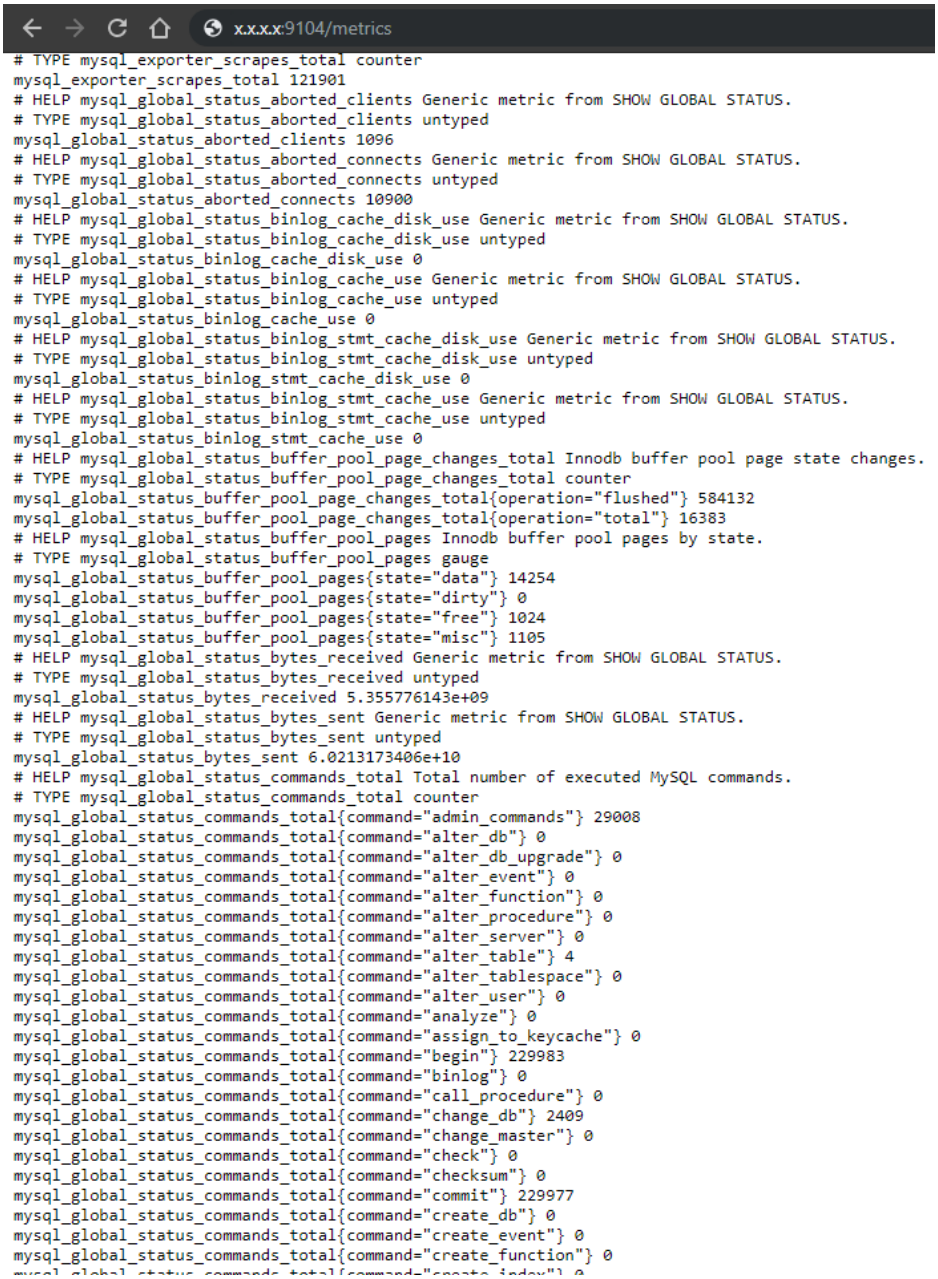
De Varnish *exporter* verzamelt de gegevens van elke backend die draait op Varnish en converteert deze naar het Prometheus formaat. Verder worden deze gegevens dan blootgesteld op http://localhost:9131/metrics waar ze worden opgehaald door Prometheus.



Figuur 54 varnish exporter metrics pagina

4.7.15 MySQLd exporter

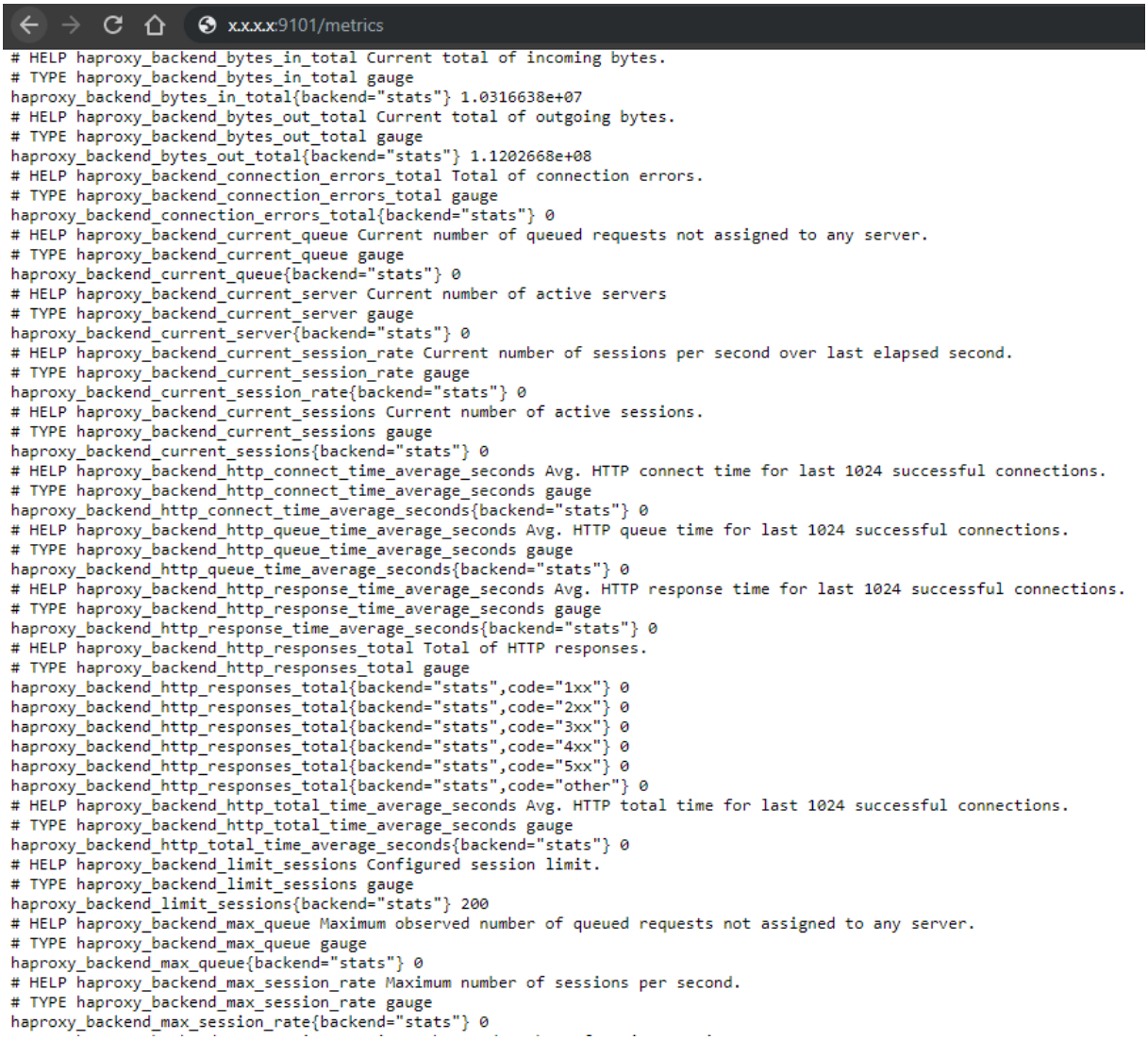
De MySQLd *exporter* wordt gebruikt voor het monitoren van een MySQL-server. De *exporter* verzamelt gegevens zoals het aantal connecties, uptime, tabellen,… . Verder wordt deze data geconverteerd naar het Prometheus formaat en wordt het blootgesteld op http://localhost:9104/metrics. Daarna worden de *metrics* opgehaald door Prometheus.



Figuur 55 MySQLd exporter metrics pagina

4.7.16 HAProxy exporter

De HAProxy *exporter* monitort de HAProxy-instantie door gebruik te maken van de *built-in* status pagina van HAProxy. Deze wordt geconverteerd van een CSV-formaat naar het Prometheus formaat en wordt dan blootgesteld op http://locahost:9101/metrics. Daarna worden de *metrics* opgehaald door Prometheus.



Figuur 56 HAProxy exporter metrics pagina

4.8 Grafana

Voor de visualisatie van de data wordt er gekozen voor Grafana. Dit is omdat Grafana zowel Elasticsearch als Prometheus ondersteunt. Verder zijn er veel bestaande dashboards die door de community zijn gemaakt voor de gebruikte *exporters* en *beats*.

Figuur 57 Grafana home dashboard

4.9 Alertmanager

Voor de *alerting* van het monitoringsysteem wordt er gebruik gemaakt van Alertmanager. De notificaties gaan zoals het vorige monitoringsysteem gebeuren via Slack.

De Alertmanager wordt geconfigureerd om alerts te posten in een Slack *channel*. De alerts worden aan de hand van een YAML-file ingeladen in Prometheus. Prometheus zal dan de query's uitvoeren en kijken of de waardes aan de voorwaarde voldoet. Na een periode die is ingesteld in de alert rule zal de alert de status "*Firing*" krijgen en wordt deze doorgestuurd naar de Alertmanager. De Alertmanager zal op zijn beurt de alert doorsturen naar Slack.

```yaml
- name: NTP
  rules:
  - alert: NTP offset
    expr: node_ntp_offset_seconds > 0.05
    for: 10s
    labels:
      severity: "critical"
    annotations:
      summary: "{{$labels.instance}} NTP offset greater then 1 second."
      description: "{{$labels.instance}} ntp offset: {{humanize $value}}."
```

Figuur 58 alert rule NTP

4.10 Prometheus API

Door middel van de HTTP API van Prometheus kan er data worden afgehaald van de Prometheus server. De API-responseformat is JSON. Met deze data zullen er grafieken worden gemaakt met PHP. De user gaat niet alle data die wordt opgehaald met de *exporters* zien, maar enkel de data dat relevant is voor hun situatie. Zo wordt de basisperformantie getoond zoals CPU, memory, disk en netwerk. Verder worden ook de processen opgehaald. Zo kan het PHP-proces van elke website getoond worden aan de corresponderende user. Hierdoor heeft de klant een beter beeld hoe het met zijn website gaat als deze op een shared hosting server staat.

http://prometheusserver.be/api/v1/query?query=(rate(namedprocess_namegroup_cpu_system_seconds_total{job="server", group="processes",groupname="php-fpm: pool user"}[5m])[30m:1m])

Deze API *call* dient voor het ophalen van een proces. Hiervoor moeten de parameters job, group en groupname worden meegeven. Job is de server waarop gezocht wordt naar het PHP-proces. Group verwijst naar de *exporter* die op de server staat. In dit voorbeeld is dit de *process exporter*. Verder is er nog de groupname. Hier wordt de naam van het process gedefinieerd. In dit voorbeeld wordt er gefilterd op naam van de user omdat de PHP-pool dezelfde naam heeft als de user. Vervolgens wordt er meegegeven dat er dertig minuten terug moet gekeken worden met een interval van één minuut.

http://prometheusserver.be/api/v1/query?query=(sum(rate(node_cpu_seconds_total{job="server", mode!="idle"}[5m])*100)[30m:30s])

Door deze API *call* kan het CPU-gebruik in seconden worden opgevraagd. Hiervoor worden de parameters job en mode meegegeven. Job is de server die wordt gemonitord. Mode is de taak waarmee de CPU bezig is. Dit kan idle, iowait, irq, nice, softirq, steal, system of user zijn. In deze situatie worden alle taken opgevraagd behalve idle. De functie sum zorgt ervoor dat de waardes over alle cores in het systeem worden opgeteld. Verder wordt er met de functie rate van de waardes een percentage gemaakt. Met deze API-call wordt de geschiedenis van de CPU tot dertig minuten terug opgevraagd met een interval van dertig seconden. De status van de CPU op het moment zelf kan worden opgevraagd door “[30m:30s]” weg te laten.

http://prometheusserver.be/api/v1/query?query=(node_memory_MemTotal_bytes{job="server"}[5m])

http://prometheusserver.be/api/v1/query?query=(node_memory_MemFree_bytes{job="server"}[5m])

http://prometheusserver.be/api/v1/query?query=(node_memory_Cached_bytes{job="server"}[5m])

http://prometheusserver.be/api/v1/query?query=(node_memory_Buffers_bytes{job="server"}[5m])

Met deze vier API *calls* kan het geheugen van een machine worden opgehaald. Om het gebruikte aantal geheugen te achterhalen moet het totale geheugen samen met het gebufferd geheugen worden afgetrokken door het vrije en gecachet geheugen.

http://185.142.227.113:8081/api/v1/query?query=(rate(node_network_transmit_bytes_total{job="server", device="eth0"}[5m])[30m:30s])

http://185.142.227.113:8081/api/v1/query?query=(rate(node_network_receive_bytes_totals{job="server", device="eth0"}[5m])[30m:30s])

Het netwerktrafiek van een server wordt via volgende twee API *calls* opgehaald. Hiervoor zijn er twee parameters nodig: de job en device. Job is de server die wordt gemonitord en device is de interface van de server die wordt gemonitord. Verder wordt de geschiedenis van dertig minuten geleden opgehaald met een interval van dertig seconden.

http://185.142.227.113:8081/api/v1/query?query=(sum(node_filesystem_avail_bytes{job="server", device!="rootfs"}))

http://185.142.227.113:8081/api/v1/query?query=(sum(node_filesystem_size_bytes{job="server", device!="rootfs"}))

Met de laatste twee API *calls* kan de grootte en de beschikbare ruimte van een disk worden opgehaald. Hiervoor zijn er twee parameters nodig, job en *device.* Job is de server die wordt gemonitord en device is de partitie. Met deze twee API *calls* wordt het percentage van de vrije opslag op die server gemaakt.

II. Onderzoekstopic

5 Onderzoeksvraag

“Wat is de beste opensourcemonitoringtool voor het monitoren van een multiserviceproviderplatform?”

Bij Level27 draaien er veel servers waar meerdere websites op draaien. Iedere website is natuurlijk anders en sommige servers vragen meer resources. In het huidige monitoringsysteem wordt enkel de server op zich gemonitord en niet de websites of applicaties.

De tools worden beoordeeld aan de hand van deelvragen:

- Wat zijn de integratiemogelijkheden in de bestaande omgeving?
- Wie heeft de duidelijkste documentatie?
- Welke tool kan de nuttigste data ophalen?
- Welke tool is het schaalbaarst?
- Wat zijn de automatisatiemogelijkheden voor elke tool?

6 Onderzoeksmethode

Het onderzoek wordt uitgevoerd in drie fases. Ten eerste is er de literatuurstudie waar er gekeken wordt naar de bestaande documentatie van de tools. Verder wordt er informatie opgezocht over de verschillende *exporters* van de verschillende tools. Uit dit rapport worden er dan twee tools gehaald.

Ten tweede wordt er een *Prove Of Concept* opgesteld met twee tools. De tools worden met elkaar vergeleken aan de hand van de deelvragen.

Ten slotte wordt er met gekozen tool een performantietest gedaan op zowel de server waar de tool op draait alsook op de server waar de *exporters* op draaien.

7 Uitwerking onderzoek

7.1 Literatuurstudie

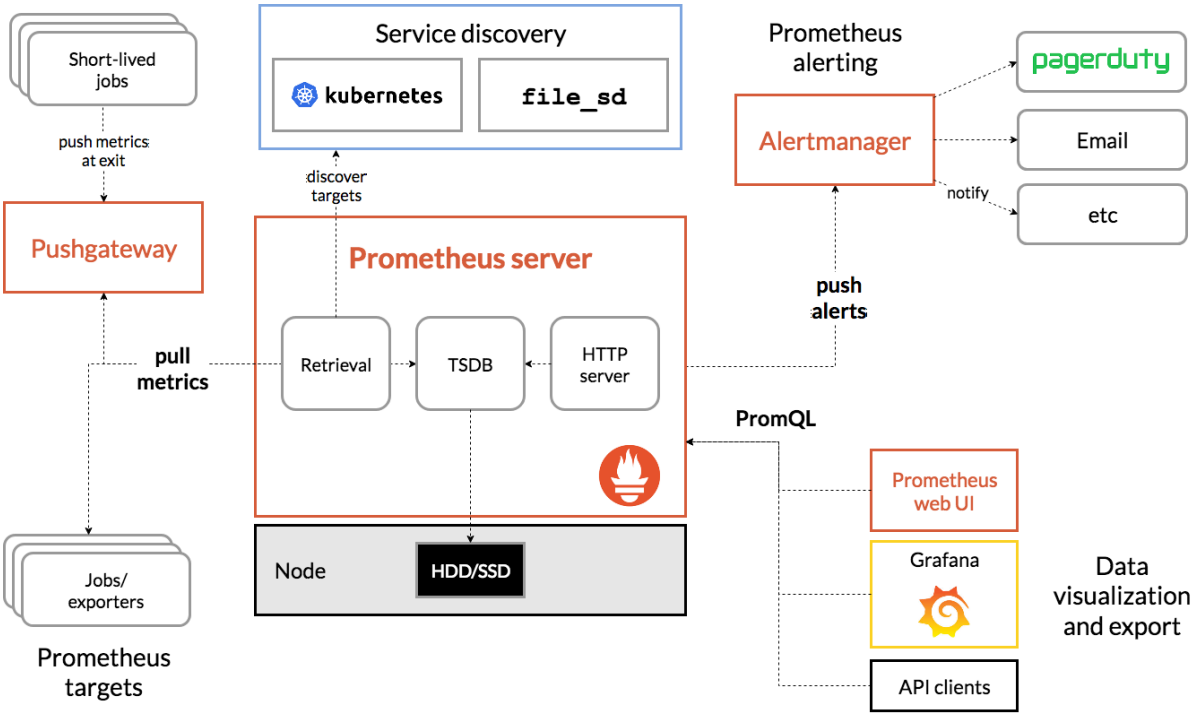
7.1.1 Push vs. pull

In een *pull based* methode zal de monitoring *agents* de target die wordt gemonitord pollen en wordt er een alarm uitgevoerd op basis van die data. Een nadeel van pull methode is dat het niet goed werkt voor *event driven time series*. Bij een pull methode kan er gemakkelijker worden gezien of een target *down* is. Ook kan er manueel via de webbrowser worden gekeken naar de gezondheid van de target [5]

In een *push based* methode wordt de data gestuurd naar de monitoring *agent.* Monitoring is gedaan door de *agent* of andere processen die de databank zal raadplegen.

7.1.2 Prometheus

Prometheus is een opensourcemonitoringsysteem en *alerting* toolkit origineel gebouwd door SoundCloud. Toen het begon in 2012 hebben veel bedrijven en organisaties aan Prometheus gewerkt. Het project had een actieve developer als user community. Nu is het een standalone opensourceproject dat onafhankelijk wordt onderhouden door verschillende bedrijven. [6]



Figuur 59 prometheus infrastructuur

Het Prometheus datamodel heeft *key-value* paren als label die tags worden genoemd. Prometheus ondersteunt *float64* datatype met gelimiteerde ondersteuning voor *string* en *timestamps* met nauwkeurigheid tot op de de miliseconde. [7]

Prometheus slaat alle data op als *time series*. *Time series* zijn streams van in tijd vermelde waardes gekoppeld aan dezelfde *metrics* en dezelfde sets van gelabelde dimensies. [7]

Binnenkomende data worden gegroepeerd in blokken van twee uur. Elk blok van twee uur bestaat uit een folder met daarin één of meerdere *chunk* files die alle *time series* data voor die tijdspanne, alsook een metadata-file en een indexfile bevatten. De indexfile is een file waarin alle *metric* namen en labels tot een *time series* wordt geïndexeerd in de *chunk* files. Wanneer er series worden gedeletet via de API, worden de te deleten records opgeslagen in *tombstone* files in plaats van de files onmiddellijk te deleten van de *chunk* files. [8]

Het blok waar er data naar binnenkomt, wordt in het werkgeheugen gehouden en is nog niet opgeslagen. De data is beveiligd tegen crashes door een *write-ahead-log* dat kan worden herhaald wanneer de server herstart na een crash. *Write-ahead-log* bestanden worden opgeslagen in de WAL folder in 128MB segmenten. Deze files bevatten ruwe data die nog niet compact is gemaakt. Hierdoor zijn deze veel groter dan normale blokken. Prometheus houdt een minimum van drie *write-ahead-log* bestanden. Servers met hoge trafiek zullen meer dan drie WAL bestanden bijhouden omdat deze ook data van twee uur terug moeten bijhouden. Het initiële blok van twee uur wordt uiteindelijk compact gemaakt in langere blokken op de achtergrond. [8]

Een limitatie van de lokale opslag is dat het niet kan worden geclusterd of gerepliceerd. Het niet duurzaam of schaalbaar wanneer er een node of harde schijf uitgaat. [8]

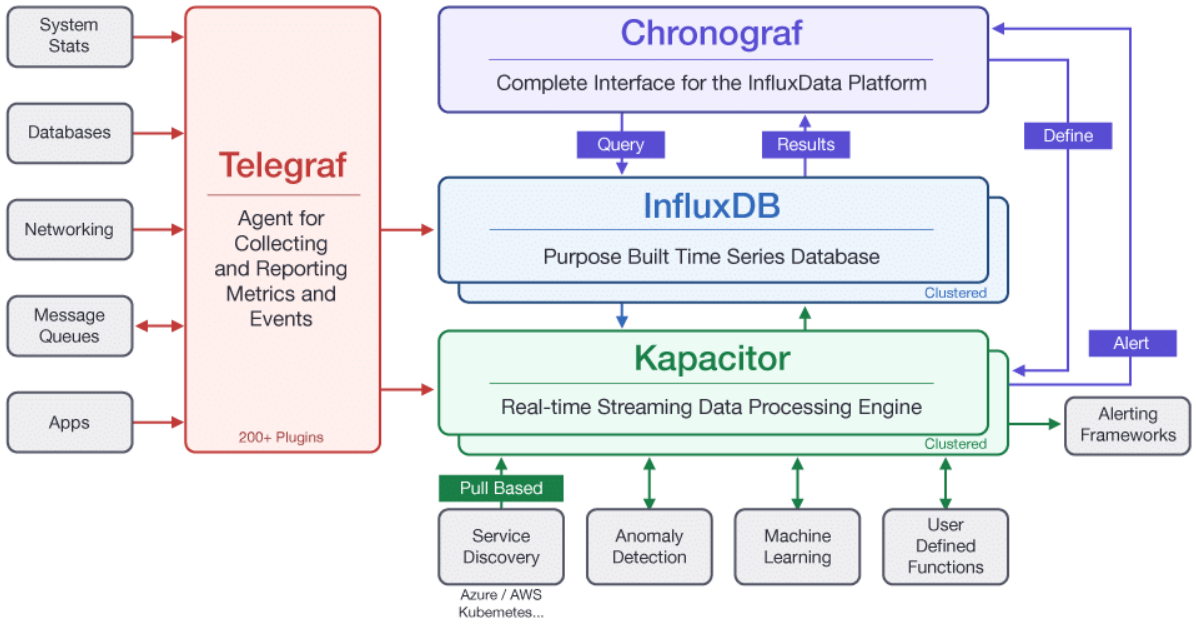
Verder heeft Prometheus een feature genaamd Federation. Federation zorgt ervoor dat een Prometheus server *time series* kan *scrapen* van een andere Prometheus server. Een *use case* hiervoor kan zijn om een schaalbare Prometheus monitoring set-up te maken of om *metrics* van een andere Prometheus te pullen naar een andere Prometheus server. [9]

Prometheus heeft een functionele *query* taal geaaemd PromQL waardoor users *time series* data in real time kunnen geselecteerd worden en verzameld. Het resultaat van een query kan worden voorgesteld als een grafiek, tabel of worden doorgestuurd naar een externe server via de HTTP API. [10]

Ten slotte heeft Prometheus nog *alerting*. *Alerting* in Prometheus wordt opgesplitst in twee delen. Ten eerste het versturen van de *Alerting* regels in Prometheus servers naar een *Alertmanager* en ten tweede het managen van die alarmen en versturen van notificaties via methodes zoals e-mail, Slack,… [11]

7.1.3 TICK Stack

TICK Stack is een samenstelling van een set van opensourceprojecten: Telegraf, InfluDB, Chronograf en Kapacitor.



Figuur 60 TICK stack infrastructuur

Telegraf

Telegraf is een *plug-in driven* server *agent* voor het verzamelen en rapporteren van *metrics*. Telegraf heeft plug-ins om een verscheidenheid aan *metrics* te kunnen afhalen van het systeem waarop het draait, kan *metrics* afhalen via *third party* API's en kan luisteren naar *metrics* via een statsd en via Kafka consumer services. Ook heeft het verschillende output plug-ins om *metrics* te versturen naar andere *datastores*, services, en bericht queues zoals: Datadog, InfluxDB, Graphite,… Ook heeft Telegraf een input plug-in voor PHP-FPM. [12]

InfluxDB

Het InfluxDB datamodel heeft key-value pairs als labels die tags worden genoemd en heeft het ook een tweede niveau van labels die *fields* worden genoemd. InfuxDB ondersteunt *timestamps* tot op de nanoseconde nauwkeurig, *float64*, *int64*, *bool*, en *string* datatypes. [13]

De InfluxDB opslag-engine heeft veel overeenkomsten met een LSM Tree. Het heeft een *write-ahead-log* en een collectie van *read-only* databestanden die een soortgelijk concept hebben als SSTables in een LSM Tree. TSM files bevatten gesorteerde en gecomprimeerde reeksen van data. [14]

InfluxDB maakt een *shard* voor elke blok tijd. Als er bijvoorbeeld een behoudpolicy met ongelimiteerde duur is, zullen de *shards* gecreëerd worden voor alle 7 dagen. Elke *shard* wordt gemapt naar een onderliggende opslag enginedatabase. Elke databank heeft een eigen WAL- en TSM-bestanden. [14]

InfluxDB heeft een query taal genaamd InfluxQL. InfluxQL is een SQL-gelijkaardige taal om interacties te kunnen doen met InfluxDB en om features te verstrekken specifiek voor het analyseren van *time series* data. [15]

Chronograf

Chronograf is InfluxData's opensourcewebapplicatie. Chronograf kan gebruikt worden met andere componenten van de TICK Stack om data te visualiseren en gemakkelijk *alerting* en automatisatie regels te maken. [16]

Kapacitor

Kapacitor is een opensourcedataverwerkingframework dat het makkelijk maakt om *alerts* te creëren, ETL jobs te starten en anomalieën te detecteren. Kapacitor kan zowel streamingdata als *batch* data verwerken. Met Kapacitor kan eigen logica of userfuncties worden ingeplugd om alerts met dynamische *thresholds* te maken. [17]

7.1.4 Nagios Core

Nagios Core een opensourcesysteem en netwerkmonitoringapplicatie. Het monitort de gespecifieerde host en services, alarmeert wanneer er dingen fout gaan en wanneer ze opnieuw beter gaan. [18]

De scope van Nagios Core is hoofdzakelijk gefocust op checks-planning, checks-uitvoering, checks verwerking, event handling en *alerting*. Maar door een grote hoeveelheid aan add-ons worden er veel features voorzien die uit de scope liggen van Nagios Core. [19]

7.1.5 Elastic Stack

Elasticsearch

Elasticsearch is een opensource en RESTful *based* zoek- en analyse-engine gebouwd op Apache Lucene. Elasticsearch is het hart van de Elastic Stack en het is de centrale opslag van de data. [20]

Logstash

Logstash is een opensourcedataverwerkingspipeline dat data binnenkrijgt van meerdere bronnen tegelijkertijd. Daarna transformeert het de data en zendt het naar Elasticsearch. [21]

Kibana

Kibana is een opensource *based* datavisualisatie- en exploratietool die gebruikt wordt voor log en *time series* analyse en applicatiemonitoring. Het biedt verschillende features aan waarmee data kunnen worden gevisualiseerd zoals histogrammen, lijndiagrammen, cirkeldiagrammen, *heat maps* en ingebouwde geolocatiesupport. [22]

Filebeat

Filebeat is een lichtgewicht *shipper* voor het forwarden en centraliseren van log data. Filebeat wordt geïnstalleerd als een *agent* op een server. Filebeat monitort de gespecificeerde logfiles, verzamelt eventlogs en forward deze naar enerzijds Elasticsearch of Logstash voor indexatie. [23]

Winlogbeat

Winlogbeat heeft 3 features: Als eerste feature kan Winlogbeat Windows eventlogs sturen naar Elasticsearch of Logstash door gebruik te maken van de Windows API's. Winlogbeat kan ook geïnstalleerd worden als een Windows-service. Ten tweede kan Winlogbeat één of meerdere eventlogs lezen door gebruik te maken van de Windows API's. Daarna worden de events gefilterd op basis van usercriteria. Ten slotte wordt de eventdata naar logstash of Elasticsearch gestuurd.

Met Winlogbeat kan eventdata verzamelen van alle eventlogs die aan het draaien zijn op een systeem. Bijvoorbeeld: applicatie-events, hardware-events, securityevents en systeemevents. [24]

Packbeat

Packetbeat is een *real-time* netwerkpakketanalysetool. Packetbeat *sniffs* de trafiek tussen servers, ontleedt de applicatielaagprotocollen en zoekt naar samenhang tussen de berichten om er een transactie van te maken. [25]

Metricbeat

Metricbeat is een lichtgewicht *shipper* dat geïnstalleerd kan worden op een server om periodiek *metrics* van besturingssystemen en actieve services te verzamelen. Metricbeat verzamelt deze *metrics* en statistieken en verstuurt deze naar Elasticsearch of Logstash. [26]

Heartbeat

Heartbeat is een lichtgewicht *daemon* dat kan geïnstalleerd worden op een remote server om periodiek te checken naar de status van services en te bepalen of de service nog beschikbaar is. Anders dan *Metric*beat die enkel zegt of de services aan of uit staan vertelt Heartbeat of de services wel bereikbaar zijn.

Heartbeat kan handig zijn om te verifiëren of de service *uptime agreement* wel gehaald wordt. Het kan ook gebruikt worden als security *use case* om te zien of bepaalde services wel van de buitenkant bereikbaar zijn. [27]

Auditbeat

Auditbeat is een lichtgewicht *shipper* dat geïnstalleerd kan worden op een server om de activiteiten van users en processen te auditen op een systeem. Bijvoorbeeld: kan auditbeat gebruikt worden om audit events van Linux Audit framework te verzamelen en te centraliseren. Auditbeat kan gebruikt worden om veranderingen in belangrijke files te detecteren. [28]

Journalbeat

Journalbeat is een lichtgewicht *shipper* voor het forwarden en centraliseren van log data van systemd "journals". Journalbeat wordt geïnstalleerd als een *agent* op een server en monitort dan de *journal* locatie die specificeert wordt, verzamelt de log events en forward deze naar Elasticsearch of Logstash. [29]

Topbeat (legacy)

Topbeat is een lichtgewicht *shipper* dat geïnstalleerd kan worden op servers om periodiek het systeemwijd en per proces CPU- en geheugenstatistieken te lezen. Daarna worden deze statistieken in Elasticsearch geïndexeerd. [30]

7.1.6 Zabbix

Zabbix is een opensourcemonitoringtool waarmee verschillende *metrics* kunnen worden verzameld. Zo kan Zabbix services, applicaties, databases, hardware, clouds, containers en virtuele machines monitoren. Verder is het ook veilig door de encryptie tussen alle Zabbix componenten en ondersteunt het ook LDAP en Active Directory. Bovendien heeft het ook een ingebouwd alarmsysteem waarmee het berichten kan versturen, ook kan het de problemen zelf oplossen. Ten slotte wordt er gebruik gemaakt van een widget gebaseerd dashboard waarmee het met behulp van grafieken, netwerkmappen,… een visueel overzicht van de IT-omgeving kan worden gemaakt. [31]

7.1.7 PRTG

PRTG is een opensourcemonitoringtool waarmee alle systemen, apparaten, trafiek en applicaties van een IT-infrastructuur kan worden gemonitord.

PRTG kan netwerksegmenten scannen door voor gedefinieerde IP-ranges te pingen. Hierdoor kan PRTG automatisch een brede range van toestellen en systemen herkennen en sensors creëren van voor gedefinieerde toesteltemplates.

Met PRTG Maps kunnen er webpagina's gemaakt worden met up-to-date monitoringdata. Ook kan er met PRTG Maps deze data publiek beschikbaar worden gemaakt.

PRTG heeft een ingebouwd alarmsysteem waarmee het notificaties kan pushen naar een smartphone. Dit gebeurt met de app die kan gedownload worden op Android, IOS en Windows Phone. Ook kunnen er notificaties worden gepusht via e-mail of SMS. Verder kan er via een API-notificatiescripts worden geschreven om een gecustomized notificatiesysteem te maken. [32]

7.2 Conclusie Literatuurstudie

De gekozen tools uit de literatuurstudie worden Prometheus en Elastic stack. Elastic stack wordt gekozen voor de grote hoeveelheden van beats en de mogelijkheid voor clustering.

Prometheus wordt gekozen voor de grote hoeveelheid *exporters*, schaalbaarheid en om het verschil te kunnen testen tussen een pull- en pushmonitoringsysteem.

7.3 Performantietesten Prometheus

Voor de performantie testen van de Prometheus server wordt er gebruik gemaakt van de tool Avalanche. Dit is een loadtester gemaakt voor Prometheus. De loadtest infrastructuur zal een lege server zijn waar er verschillende instanties van Avalanche op zal draaien aan de hand van het aantal exporters een type server nodig heeft.

Er zijn verschillende types van servers die er draaien bij Level27. In deze situatie zullen de databaserver, *shared* webserver en een Docker-server worden nagebootst. Voor de Windows server zal er geen gebruik gemaakt worden van Avalanche omdat deze niet *compatible* is met Windows, maar zullen de *exporters* worden gebruikt.

Type server	**Aantal *exporters***
Webserver	5
Shared webserver	6
Databaseserver	3
Windows server	2
Docker-server	5

Tabel 1 type server aantal exporters

Om tot een reëel beeld te krijgen hoeveel data elke *exporter* verzameld, wordt er gekeken naar de bestaande infrastructuur van Level27. Hierbij zal er een gemiddelde worden gemaakt van het aantal PHP *pools* op een *shared* webserver en wordt voor elke website op de server één database gerekend.

Gemiddeld	**Gemiddeld exclusief uitschieters**
36,71	31,1

Tabel 2 gemiddeld aantal websites shared hosting server

Voor de *shared* webserver zullen er éénendertig PHP *pools* worden nagebootst met elk één database.

***Shared* webserver**			
Exporter	**Website/database *timeseries***	**Standaard *timeseries***	***Aantal timeseries /exporter***
PHP-FPM *exporter*	22	/	682
Process *exporter*	22	176	858
Node *exporter*	/	495	495
Apache *exporter*	/	24	24
Nginx *exporter*	/	9	9
Mysql *exporter*	/	917	917
Totaal	44	1621	2985

Tabel 3 aantal timeseries shared webserver

Databaseserver	
Exporter	***Aantal timeseries/exporter***
Node *exporter*	495
Process *exporter*	176
Mysql *exporter*	917
Totaal	1588

Tabel 4 aantal timeseries databaseserver

Het aantal website op een webserver varieert van klant tot klant. In dit voorbeeld wordt er geschat op drie websites per webserver.

Webserver			
Exporter	**Website *timeseries***	**Standaard *timeseries***	***Aantal timeseries /exporter***
Node *exporter*	/	495	495
Process *exporter*	22	176	242
Apache *exporter*	/	24	24
Nginx *exporter*	/	9	9
PHP-FPM	22	/	66
Totaal	44	704	836

Tabel 5 aantal timeseries webserver

Het aantal containers op een Docker-server varieert van klant tot klant. In dit voorbeeld wordt er geschat op vier containers per server

Docker-server			
Exporter	**Container *timeseries***	**Standaard *timeseries***	**Aantal *timeseries /exporter***
Node *exporter*	/	495	495
Process *exporter*	/	176	176
Apache *exporter*	/	24	24
Nginx *exporter*	/	9	9
Container *exporter*	540	/	2160
Totaal	540	704	2864

Tabel 6 aantal timeseries Docker-server

Windows-server	
Exporter	**Aantal *timeseries/exporter***
WMI *exporter*	9086
MSSQL *exporter*	21
Totaal	9107

Tabel 7 aantal timeseries Windows-server

De waardes in de tabellen zijn het gemiddelde gebruikspercentage van dertig minuten.

Prometheus-server (Avalanche)		
***Scrape* target**	**CPU**	**MEM**
/	1.33%	11.39%
Shared webserver	7.47%	31%
Webserver	3,36%	15,42%
Databaseserver	3,25%	16.63%
Docker-server	2.02%	24.04%

Tabel 8 Prometheus-server performance scraping Avalanche

Target server performantie (Avalanche)		
Server	**CPU**	**MEM**
/	0.4%	1.03%
Shared webserver	1.48%	7,12%
Webserver	1.32%	7,08%
Databaseserver	0.68%	6.59%
Docker-server	0.48%	7.20%

Tabel 9 Target server performantie Avalanche

Prometheus-server (exporters)		
***Scrape* target**	**CPU**	**MEM**
/	1.33%	11.39%
Shared webserver	/	/
Webserver	3,36%	15,42%
Databaseserver	2,25%	13.61%
Docker-server	3.47%	16.15%
Windows-server	1.40%	14.21%

Tabel 10 Prometheus-server performance exporters

Target server performantie (Windows exporters)		
Server	**CPU**	**MEM**
/	18%	80%
Windows-server	19%	80%

Tabel 11 Target server performantie Windows exporters

Target server performantie (Linux exporters)		
Server	**CPU**	**MEM**
/	7.23%	66.88%
Shared webserver	/	/
Webserver	8.36%	68.13%
Databaseserver	8.12%	67.42%
Docker-server	8.56%	68.78%

Tabel 12 target server performantie Linux exporters

Bij het ophalen van de data van een *shared* webserver (Avalanche) waarbij elke *metric* één timeseries heeft zal de Prometheus-server een out of memory error krijgen na één uur en dertig minuten. Doordat er veel *metrics* zijn met elk één *timeseries* loopt de *memory* vol. De Prometheus server heeft twee CPU's en 4GB RAM. Wanneer er meerdere timeseries onder één *metric* worden geplaatst kan Prometheus dit makkelijker aan. De *exporters* die worden gebruikt sluiten meer aan bij de tweede situatie. De *exporters* hebben amper tot geen impact op de performantie van de *target* server.

7.4 Performantietesten ELK stack

Voor het testen van de performantie van de ELK stack word er gebruik gemaakt van Metricbeat. Om tot een reële situatie te komen zal Metricbeat gebruik maken van de Apache, Nginx, MySQL, system en PHP-FPM modules. De ELK stack server heeft 4 CPU's en 8GB memory.

ELK stack performantie		
beats	**CPU**	**MEM**
/	2.59%	48.95%
Metricbeat	3.57%	49.68%

Tabel 13 ELK stack performantie test

Target server performantie (Metricbeat)		
beats	**CPU**	**MEM**
/	8.95%	57.69%
Metricbeat	9.48%	58.32%

Tabel 14 ELK stack target server performantie

ELK stack gebruikt veel RAM-geheugen zonder dat er een Metricbeat data *pushed* naar elasticsearch. De impact van Metricbeat op de performantie van ELK stack alsook de target server is klein.

8 Conclusie onderzoek

8.1 Wat zijn de integratiemogelijkheden in de bestaande omgeving?

Prometheus heeft een *exporter* die informatie verzamelt van Nagios en de checks van Nagios uitvoert. Het enige wat de Nagios core moet ondersteunen is de NEB interface. Voor de Elastic stack is er geen integratie met Nagios maar wel een Nagios plug-in die hetzelfde doet als Nagios.

Prometheus heeft geen *exporter* om de *metrics* van Elasticsearch door te sturen. Wel heeft het een *exporter* die Elasticsearch zelf kan monitoren.

Voor een integratiemogelijkheid met het controlepaneel heeft Prometheus een API waarmee *metrics* kunnen worden afgehaald. De data is in JSON-formaat en kan met PHP worden omgezet naar grafieken. [33]

8.2 Wie heeft duidelijkste documentatie?

Het grote verschil tussen Elastic stack en Prometheus is dat Elastic stack en zijn beats beheerd worden door een bedrijf. Prometheus aan de andere kant wordt ook beheerd door een bedrijf maar het merendeel van de *exporters* wordt gemaakt/beheerd door de community. Dit zorgt ervoor dat de configuratie niet altijd volledig is of gewoon fout. De documentatie van Elastic stack staat volledig uitgeschreven voor elke *build* versie van een service. Bij Prometheus is dit ook, maar niet voor de *exporters*. Voor de *exporters* bestaat er enkel documentatie voor de laatste nieuwe versie .

8.3 Welke tool kan de nuttigste data ophalen?

Elastic stack kan veel data ophalen door de verschillende beats die ze hebben. Maar niet alle data die ze kunnen ophalen is nuttig voor deze situatie. In deze opdracht wordt er vooral gekeken naar het verzamelen van *metrics*. Deze worden opgehaald via Metricbeat. Metricbeat heeft een grote verzameling van modules waarmee het data kan afhalen. Alleen is deze niet uitgebreid genoeg om aan de node te voldoen.

Prometheus is enkel gericht op het ophalen van *metrics* en doordat veel van de *exporters* gemaakt worden door de community, heeft het een veel groter aanbod van *exporters* om services te monitoren. Wat Prometheus wel niet kan, is het verzamelen van logs. Hier blinkt de Elastic stack dan weer wel uit. Indien er latere uitbreidingen voor een centralisatie van de logs zou gebeuren, zal er dus een Elastic stack moeten worden opgezet.

8.4 Welke tool is het schaalbaarst?

Prometheus en Elasticsearch zijn allebei schaalbaar. De *exporters* en beats kunnen automatisch worden gedeployd bij het aanmaken van een nieuwe server. Maar doordat Prometheus een *pull* monitoringsysteem is, moet er voor elke nieuwe *exporte*r een aanpassing gedaan worden in de configuratiefile van Prometheus.

Met Prometheus kan een *master-slave* netwerk worden opgezet via de Federation functie. De master *node* gaat dan *time series scrapen* van andere Prometheus servers. En de *slave nodes* gaat *metrics* afhalen van de *target* servers.

Elasticsearch heeft een uitgebreidere clusterfunctie. De kracht van de Elasticsearch cluster ligt bij de distributie van taken, zoeken en indexering over al de *nodes* in de cluster.

De *nodes* krijgen allemaal verschillende jobs of verantwoordelijkheden. Als eerste zijn er de data *nodes*. Deze zorgen voor het opslaan van de data en voor het *queryen* en de aggregatie van de data. Als tweede zijn er de master *nodes*. Deze zorgen voor het managen van de cluster en configuratieacties zoals het toevoegen of verwijderen van *nodes*. Als derde zijn er de client *nodes*. Deze *nodes* zorgen voor het *forwarden* van clusterverzoeken naar de master *nodes* en datagerelateerde verzoeken naar de data *nodes*. Als laatste zijn er de Ingest *nodes*. Deze zorgen voor het voorverwerken van documenten voor indexering. [34]

8.5 Wat zijn de automatisatie mogelijkheden voor elke tool?

Als eerste mogelijkheid kan Elasticsearch worden opgezet met Docker. Ten tweede bestaan er voor Elasticsearch Chef *cookbooks* om automatisch een Elasticsearch node of cluster te maken. Ook zijn er guides hoe je zo een *cookbook* maakt.

Prometheus kan ook worden opgezet met Docker alsook zijn er *cookbooks* van Chef voor zowel Prometheus alsook de Alertmanager.

Voor automatisatie van Metricbeat kan deze worden erbij geïnstalleerd bij het maken van een nieuwe server met Chef.

Verder bestaan er al Chef *cookbooks* voor bepaalde *exporters*, maar die niet allemaal nodig zijn voor deze situatie.

8.6 Vergelijkingsmatrix

Key values/tool	Prometheus	ELK stack
Integratie		
Documentatie		
Data		
Schaalbaarheid		
Automatisatie		
Performantie		

Prometheus scoort beter op data dan ELK stack omdat het veel meer data kan ophalen met de verschillende *exporters* dan ELK stack met Metricbeat. Ook scoort Prometheus beter op performantie omdat ELK stack *Idle* meer RAM-geheugen gebruikt dan Prometheus. Dit heeft te maken met de heap-size die wordt ingesteld op ELK stack. Ten slotte scoort ELK stack beter op documentatie omwille van de duidelijkheid. Prometheus wordt gekozen als tool om verder uit te werken. Dit is omdat data en performantie harder doorwegen dan documentatie.

Reflectie

Tijdens mijn stage bij Level27 ben ik op professioneel vlak gegroeid. Door mijn stage gedaan te hebben in de IT-support wereld, heeft het ervoor gezorgd dat ik veel heb bijgeleerd hoe je met klanten omgaat. Verder hebben de securitymaatregelen die ik tijdens mijn stageopdracht ben tegen gekomen, mij nog bewuster gemaakt van het belang aan security. Vervolgens heb ik nog veel bijgeleerd over verschillende tools die ik tijdens mijn stageopdracht ben tegengekomen zoals Nginx, NTP, Solr, Varnish,…

Het begin van mijn stage was voornamelijk stressvol, toen ik realiseerde hoe groot de opdracht eigenlijk was. Dit ben ik tegen gegaan door de opdracht op te delen in kleinere taken en er sprints van te maken. Hierdoor leek de opdracht minder groot. Dit heeft me heel hard geholpen om alles op tijd af te werken en mijn hoofd koel te houden. Verder moest ik elke week een update geven aan mijn technische begeleider. Dit vond ik wel goed, daar ik wat druk nodig heb omdat ik vaak dingen uitstel. In de toekomst, wanneer ik niemand heb die mij controleert, ga ik proberen mezelf deadlines op te leggen. Ik weet dat dit een werkpunt voor mezelf is en ik probeer deze uitdaging ook aan te gaan. Verder heb ik mezelf nog gepushed om met Linux te werken in plaats van met Windows, door alles op Linux te maken. Ik heb ervoor gekozen om met Linux te werken omdat mijn basis van Linux minder goed is dan die van Windows. Hierdoor heb ik veel bijgeleerd over Linux wat ik anders nooit zou hebben gedaan.

Ook heb ik vanuit Level27 de kans gekregen om van het bedrijfsleven te proeven. Zo heb ik doorheen mijn stage mogen helpen bij de support. Hier kwamen de soft skills die ik tijdens mijn opleiding heb aangeleerd van pas. Het afhandelen van telefoongesprekken was in het begin meestal nog stressvol voor mij. Dit ben ik tegengegaan door altijd eerst twee seconden te wachten en dan pas de hoorn op te nemen. Hierdoor was ik altijd veel kalmer aan de telefoon en versprak ik me minder. Ook heb ik gewerkt aan een mailmigratieproject waarbij er mailboxen moesten gemigreerd worden van Gmail en Kerio naar Office365. Hier had ik veel handmatig werk voor niks gedaan omdat ik er te snel aan wou beginnen. Volgende keer als ik aan een groter project begin, kan ik beter alle opties die er zijn overlopen in plaats van de eerste de beste te nemen. Dus eerst mijn keuze goed overwegen en dan beslissen.

Mijn persoonlijke reflectie over deze stage is over het algemeen heel positief. De stageopdracht was heel interessant en leerrijk. Ook is het leuk om weten dat mijn werk later geïmplementeerd wordt.

Bibliografie

[1] P. Fastré, „wiki level27,” Level27, 1 Januari 2019. [Online]. Available: wiki.level27.be. [Geopend 8 Maart 2019].

[2] P.-H. Kamp, „varnish-cache,” Varnish, 1 Januari 2017. [Online]. Available: https://varnish-cache.org/intro/index.html#intro. [Geopend 13 Mei 2019].

[3] Memcached, „memcached,” memcached, 27 April 2019. [Online]. Available: https://memcached.org/. [Geopend 13 Mei 2019].

[4] Elastic, „www.elastic.co,” Elasticsearch, 2019. [Online]. Available: https://www.elastic.co/guide/en/elasticsearch/reference/current/node.name.html. [Geopend 15 03 2019].

[5] Prometheus, „www.prometheus.io,” Prometheus, 2019. [Online]. Available: https://prometheus.io/docs/introduction/faq/#why-do-you-pull-rather-than-push. [Geopend 8 Maart 2019].

[6] Prometheus, „ww.prometheus.io,” Prometheus, 2019. [Online]. Available: https://prometheus.io/docs/introduction/comparison/. [Geopend 03 Maart 2019].

[7] Prometheus, „www.prometheus.io,” Prometheus, 2019. [Online]. Available: https://prometheus.io/docs/introduction/overview/. [Geopend 8 maart 2019].

[8] Prometheus, „www.prometheus.io,” Prometheus, 2019. [Online]. Available: https://prometheus.io/docs/prometheus/latest/storage/. [Geopend 8 Maart 2019].

[9] Prometheus, „www.prometheus.io,” Prometheus, 2019. [Online]. Available: https://prometheus.io/docs/prometheus/latest/querying/basics/. [Geopend 8 Maart 2019].

[10] Prometheus, „www.prometheus.io,” Prometheus, 2019. [Online]. Available: https://prometheus.io/docs/prometheus/latest/federation/. [Geopend 8 Maart 2019].

[11] Prometheus, „www.prometheus.io,” Prometheus, 2019. [Online]. Available: https://prometheus.io/docs/alerting/overview/. [Geopend 8 Maart 2019].

[12] influxdata, „www.influxdata.com,” Influxdata, 2019. [Online]. Available: https://www.influxdata.com/time-series-platform/telegraf/. [Geopend 8 Maart 2019].

[13] Prometheus, „www.prometheus.io,” Prometheus, 2019. [Online]. Available: https://prometheus.io/docs/introduction/comparison/. [Geopend 8 Maart 2019].

[14] Influxdata, „docs.influxdata.com,” Influxdata, 2019. [Online]. Available: https://docs.influxdata.com/influxdb/v1.7/concepts/storage_engine/. [Geopend 8 Maart 2019].

[15] Influxdata, „www.influxdata.com,” Influxdata, 2019. [Online]. Available: https://www.influxdata.com/time-series-platform/influxdb/. [Geopend 8 Maart 2019].

[16] Influxdata, „www.docs.influxdata.com," Influxdata, 2019. [Online]. Available: https://docs.influxdata.com/chronograf/v1.7/. [Geopend 8 Maart 2019].

[17] Influxdata, „www.docs.influxdata.com," Influxdata, 2019. [Online]. Available: https://docs.influxdata.com/kapacitor/v1.5/. [Geopend 8 Maart 2019].

[18] Nagios, „www.assets.nagios.com," Nagios, 2019. [Online]. Available: https://assets.nagios.com/downloads/nagioscore/docs/nagioscore/4/en/about.html#whatis. [Geopend 8 Maart 2019].

[19] Nagios, „www.nagios.org," Nagios, 2019. [Online]. Available: https://www.nagios.org/projects/nagios-core/. [Geopend 8 Maart 2019].

[20] Elasticsearch, „www.elastic.co," Elasticsearch, 2019. [Online]. Available: https://www.elastic.co/products/elasticsearch. [Geopend 8 Maart 2019].

[21] Elastic, „www.elastic.co," Elasticsearch, 2019. [Online]. Available: https://www.elastic.co/products/logstash. [Geopend 8 Maart 2019].

[22] Amamzon, „www.aws.amazon.com," Amamzon, 2019. [Online]. Available: https://aws.amazon.com/elasticsearch-service/kibana/. [Geopend 8 Maart 2019].

[23] Elastic, „www.elastic.co," Elastic, 2019. [Online]. Available: https://www.elastic.co/guide/en/beats/filebeat/current/filebeat-overview.html. [Geopend 8 Maart 2019].

[24] Elastic, „www.elastic.co," Elastic, 2019. [Online]. Available: https://www.elastic.co/guide/en/beats/winlogbeat/current/_winlogbeat_overview.html. [Geopend 8 Maart 2019].

[25] Elastic, „www.elastic.co," Elastic, 2019. [Online]. Available: https://www.elastic.co/guide/en/beats/packetbeat/current/packetbeat-overview.html. [Geopend 8 Maart 2019].

[26] Elastic, „www.Elastic.co," Elastic, 2019. [Online]. Available: https://www.elastic.co/guide/en/beats/metricbeat/current/metricbeat-overview.html. [Geopend 8 Maart 2019].

[27] Elastic, „www.elastic.co," Elastic, 2019. [Online]. Available: https://www.elastic.co/guide/en/beats/heartbeat/current/heartbeat-overview.html. [Geopend 8 Maart 2019].

[28] Elastic, „www.elastic.co," Elastic, 2019. [Online]. Available: https://www.elastic.co/guide/en/beats/auditbeat/current/auditbeat-overview.html. [Geopend 8 Maart 2019].

[29] Elastic, „www.elastic.co," Elastic, 2019. [Online]. Available: https://www.elastic.co/guide/en/beats/journalbeat/current/journalbeat-overview.html. [Geopend 8 Maart 2019].

[30] Elastic, „www.elastic.co,” Elastic, 2019. [Online]. Available: https://www.elastic.co/guide/en/beats/topbeat/current/_overview.html. [Geopend 8 maart 2019].

[31] Zabbix, „www.zabbix.com,” Zabbix, 2019. [Online]. Available: https://www.zabbix.com/features. [Geopend 8 Maart 2019].

[32] PRTG, „www.paessler.com,” Paessler, 2019. [Online]. Available: https://www.paessler.com/prtg. [Geopend 8 Mei 2019].

[33] Prometheus, „www.prometheus.io,” Prometheus, 24 April 2019. [Online]. Available: https://prometheus.io/docs/prometheus/latest/querying/api/. [Geopend 7 Mei 2019].

[34] D. Berman, „logz.io,” logz.io, 14 Februari 2018. [Online]. Available: https://logz.io/blog/elasticsearch-cluster-tutorial/. [Geopend 23 April 2019].