Proof of Concept: Logisland

Realisatie

Bachelor in de toegepaste informatica

Academiejaar 2018-2019

Campus Geel, Kleinhoefstraat 4, BE-2440 Geel

Hanot Lukas

LID VAN ASSOCIATIE KU LEUVEN

THOMAS MORE

INHOUDSTAFEL

INLEIDING....4

1 DE BASIS....5

2 INUITS 6

3 POC – LOGISLAND....7

3.1 Inleiding....7
3.2 Fasering....7
3.3 Hoe werkt Logisland....7
3.4 Lokale testomgeving....7
3.5 Rebootstrappen van Lisbon.hypprod.inuits.eu....8
3.6 Installatie van logisland in Logisland.playground.inuits.eu....15
3.6.1 Vereisten....16
3.6.2 Start docker....16
3.6.3 Configuratie van Logisland....17
3.6.4 Logstash....26
3.6.5 Link Kibana....27
3.7 Hoe werkt Logisland?....28
3.7.1 The Engine....28
3.7.2 The Controller....29
3.7.3 The Streams....31
3.7.4 The Indexer....33
3.7.5 Managementcommands....34
3.7.6 Notes....35
3.7.7 Spark stream processing examples....37
3.7.8 Processors....40

4 CONCLUSIE....45

5 BRONNEN....46

INLEIDING

“Wat in godsnaam is Logisland” was de vraag die ik stelde bij de start van mijn stage. Een nieuwe tool (1), een nieuwe manier van werken (2) en dit in een onbekende omgeving (3): dat alles was enorm overweldigend bij het begin van de stage. Drie maanden later kan ik, met opgeheven hoofd, mijn verworven kennis delen via dit realisatiedocument.

Om de lezer een beter inzicht te geven in mijn kennis toont het eerste hoofdstuk ‘**De basis’** mijn ervaringen met Linux en Open Source software en ga ik na hoe dit opweegt tegenover de vereisten van Inuits. In het hoofdstuk erna verdiep ik mij in **de werkwijze van Inuits**: de technologieën, de software en de samenwerking. Dit wordt natuurlijk gevolgd door de **Proof of Concept** en wordt afgesloten met mijn **bevindingen en conclusies** over de al dan niet bruikbaarheid van Logisland binnen Inuits.

Logisland is een event minent schaalbaar platform dat gericht is op een grote doorvloed van events. Deze zin zal sommigen een idee geven van wat Logisland is en andere eens aan hun hoofd doen krabben. Daarom staat in het hoofdstuk **Logisland** een duidelijkere/ uitgebreidere uitleg waarmee u, de lezer, hopelijk een inzicht krijgt in deze tool.

Voor mij was de tool Logisland opzetten DE grote uitdaging maar het eigenlijke doel van de stage was meer dan dat. Mijn opdracht was namelijk uittesten of deze tool een aanwinst zou zijn voor Inuits. Dus na het opzetten van de tool, moest ik de tool analyseren. Dit document eindigt dan ook met een analyse van de tool en de redenen waarom het wel of niet een aanwinst zou zijn voor Inuits. De conclusie is een samenvatting van zowel mijn ervaring met Logisland als de mening van Kris, mijn stagementor.

Ik wil hier graag een aantal mensen bedanken die mij de kans hebben gegeven tot deze fantastische ervaring. Eerst en vooral dank ik mijn ouders ,die mij alle jaren van mijn studies hebben gesteund. Natuurlijk dank ik ook de hogescholen, waar ik mijn opleiding heb gevolgd zowel Karel de Grote te Antwerpen als Thomas More Geel. Deze laatste hogeschool heeft mij met open armen ontvangen en heeft mij een boeiend en leerrijk laatstejaar gegeven. Dit werk zou nooit tot stand gekomen zijn zonder mijn stagebedrijf Inuits , waar ik massa’s ervaring heb opgedaan en dat mijn interesse in Open Source, zowel software als community, extra hard heeft aangewakkerd. Rest mij nog specifiek mijn stagementor Kris Buytaert te danken, die mij veel heeft bijgeleerd en die ik hoop nog vaak tegen te komen tijdens Linux evenementen. Ook richt ik een woord van dank aan Bart Portier, mijn stagebegeleider en docent aan Thomas More, die mij streng maar rechtvaardig het juiste pad heeft getoond tijdens deze stage en de documentatie hiervan. En dan rest er nog 1 persoon, last but not least, mijn vriendin Melissa, die mij ondanks de stressvolle periode, door dik en dun heeft gesteund.

1 De basis

Een stage doen in een bedrijf dat zich focust op Open Source kan niet anders dan met de nodige voorkennis van het Linux besturingssysteem. Ik ben voor het eerst Linux beginnen gebruiken eerder uit noodzaak. Tijdens mijn stage in het middelbaar heb ik een pc gemaakt uit reserveonderdelen, die nog in het stagebedrijf lagen en op deze pc miste nog een besturingssyteem. Aangezien ik deze pc maximum een maandje ging gebruiken zou een nieuw Windowspakket (te betalen) installeren op dit systeem, nogal getuigen van verspilling. Daarom koos ik om er Ubuntu op te zetten. Zo is Linux hierdoor al ongeveer 6 jaar mijn daily-driver. Linux is mijn go-to-oplossing voor al mijn besturingssystemen van Raspberry Pi tot de server, die ik thuis draai. Graag wil ik mij dan ook steeds meer verdiepen in deze taal en ga daaarom steevast elk jaar naar Fosdem (ULB Brussel) en daar is dit jaar het event ConfMgmtCamp en Loadays bijgekomen. Op aanraden ben ik ook lid van een hackerspace te Deurne, waar zo goed als iedereen Linux gebruikt op zijn machines en in projecten.

Zoals hierboven vermeld, werk ik zelf al 6 jaar met Linux als mijn standaard daily-driver. Dit had natuurlijk een enorme invloed op de keuze van mijn stagebedrijf. De voorbije jaren heb ik amper kans gehad om zelf projecten in Linux te creëren dus zocht dit keer gericht naar een bedrijf waar Linux de standaard is. Dat Inuits mij accepteerde als stagiair, maakte mij terecht blij. Door mijn interesse en ervaring met Linux, had ik natuurlijk al een streepje voor. Dit bedrijf zocht studenten met zowel een basiskennis als een grote voorliefde voor de Linux-tools in het bijzonder.

Natuurlijk kende ik niet alle tools, die Inuits gebruikt. Dus dat alleen vond ik al fantastisch: van sommige tools had ik nog nooit gehoord of had ik geen idee van hun bestaan. Het werken met pipelines, die automatie mogelijk maken zoals Jenkins en Puppet, was een volledig nieuwe ervaring voor mij, die mij pas echt een inzicht gaf op de werkwijze binnen bedrijven. De verschillende communicatieplatformen zoals Rocket.chat en Zimbra, die volwaardige alternatieven zijn op gesloten commerciële tools zoals exchange en skype, waren leuk om te ontdekken. En vooral de ethos om met en aan opensource te werken en deze te verbeteren door actief deel te nemen aan deze cultuur, waren een ware oogopener voor mij en een doel om naar te streven: hier wil ik aan meewerken!

De grootste uitdaging was natuurlijk om tijdens deze stage de werkwijze te leren van Inuits. Hoe doe ik juist aanpassingen aan de infrastructuur, hoe zorg ik dat dit geautomatiseerd wordt, hoe monitor ik deze wijzigingen en waarom vragen mensen mij om problemen te ack'en (verwijzing 1)?

2 INUITS

Inuits is opgericht in 2007. Dit betekent dat het bedrijf ondertussen toch al 12 jaar van IT-evolutie heeft meegemaakt waardoor er een groot netwerk van virtuele server, verschillende environments en tools is ontstaan. Dit maakt het voor nieuwe werknemers en stagiairs moeilijk om snel een grip te krijgen op de omvang en complexiteit van het netwerk. Daarnaast bezit Inuits zelf geen fysieke infrastructuur: deze bevindt zich allemaal op een cloud-server bij andere providers zoals OVH.

Dit netwerk onderhouden door op elke afzonderlijke machine settings te gaan aan- en uitzetten, is dan ook geen optie. De logische oplossing is dan ook automatisatie: hiervoor gebruikt Inuits een samenwerking tussen Redmine, Jenkins en Puppet.

Redmine	Een web-gebaseerde project manager en issue-tracker-applicatie. Te vergelijken met bijvoorbeeld Jira, Sourceforge en Microsoft Project.
Jenkins	Een automatisatieserver die de non-human delen van software-development op zich neemt en continuous-delivery mogelijk maakt. Vergelijkbare voorbeelden zijn Jetbrains Teamcity, Azure DevOps en Travis CI.
Puppet	Een software configuration- management-tool, gebaseerd op een client-server model. De clients halen hun configuratie op bij de server waarna de puppet-client de machine configureert zoals aangegeven op de server. Vergelijkbare voorbeelden zijn Ansible, Chef en System Center Configuration Manager

Wanneer een administrator een aanpassing wil doen aan de configuratie van een node zoals bijvoorbeeld kibana.playground.inuits.eu, clonet de gebruiker de puppet-configuratie vanaf Redmine. Voert daarna de nodige aanpassingen uit in zijn locale repository en commit deze dan terug naar Redmine. Jenkins merkt automatisch op dat er een wijziging is gebeurd aan het bestand en voegt deze toe aan de queue. Als de wijziging alle Jenkins-tests doorstaat, wordt de wijzing uitgevoerd en de actieve puppet-configuratie aangepast. Om de 30 minuten vindt er een puppet-run plaats op de client waardoor de puppet-client bij de server gaat controleren of er wijzigingen moeten worden doorgevoerd en dus de laatste configuratie wordt toegepast.

3 POC – LOGISLAND

3.1 Inleiding

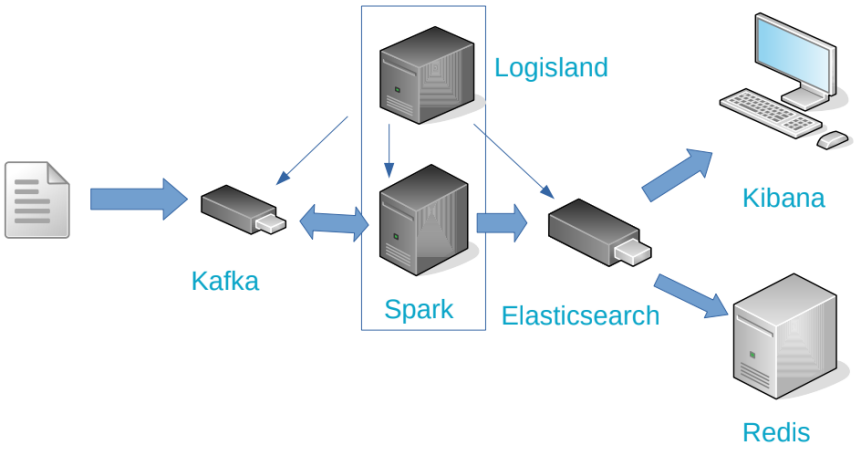
Logisland is een applicatie ontwikkeld om events te verzamelen en verwerken. Onder events verstaan we bijvoorbeeld logs, kliks op een webpagina, alerts van andere diensten, … met andere woorden alle gebeurtenissen waar we een tijdwaarde aan kunnen bevestigen. Inuits wil weten of deze tool een meerwaarde kan opleveren aan de logs, die ze momenteel al verzamelen en dan voornamelijk of ze op deze events kunnen zoeken naar sequentiële patronen, frequente patronen en de correlaties tussen tijdseries en gebeurtenissen.

3.2 Fasering

Mijn Fasering kan u terugvinden in mijn Plan van Aanpak maar zit ook toegevoegd bij de bronnen. Tijdens mijn stage heb ik mij heel goed kunnen houden aan mijn vooropgestelde planning en ik ben mezelf heel dankbaar voor de extra tijd, die ik telkens had ingeplant, want deze was nodig. Elke stap vooruit in dit project bracht een karrevracht nieuwe opdrachten met zich mee. Desondanks ben ik erin geslaagd om mijn doelen te bereiken binnen de planning.

3.3 Hoe werkt Logisland

Logisland is eigenlijk een uitbreiding op de ELK stack wat dus zorgt voor een grote complexiteit. Ik zal proberen om de uitleg zo simpel mogelijk te houden maar let voornamelijk op de illustratie hieronder om het overzicht te bewaren. Laten we van links naar rechts gaan en zo de datastroom volgen. Aan de linkerkant beginnen we met de Logstash server: deze ontvangt logs en andere records en filtert deze. De gefilterde records worden dan doorgestuurd naar het Kafka topic, waar deze worden opgevangen en het “Logisland systeem” betreden. De records wachten in de Kafka-server tot deze door spark verwerkt kunnen worden naar events. Wanneer één spark verwerking klaar is kan deze of terug naar Kafka gestuurd worden of naar Elasticsearch/redis weggeschreven worden. Hierdoor kan data verwerkt worden en dan op de verwerkte data verdere verwerkingen verricht worden. Als alle spark-operaties verricht zijn wordt de data weggeschreven naar Elasticsearch en/of redis. In Elasticsearch wordt data bijgehouden in een actieve staat deze is dus snel doorzoekbaar maar minder stabiel terwijl redis dient voor stabiele opslag. De laatste stap is Kibana: deze leest de events die zich in Elasticsearch bevinden en daardoor kan de gebruiker de events bekijken en in grafieken gieten om deze te bestuderen.



3.4 Lokale testomgeving

Elk project moet ergens beginnen en dat van mij begon, nadat ik de werkwijze van Inuits beter begreep. Hiermee kende ik de infrastructuur nog niet maar dat was voor stap 1 nog geen probleem: het opzetten van de tool moest eerst op mijn eigen laptop. Logisland is terug te vinden op de Hurence Github[1] pagina en bezit online documentatie[2].

Het was al snel duidelijk dat de docker-setup veel makkelijker zou verlopen dan zelf een hele ELK [3]stack opzetten. Docker opzetten in een Centos omgeving is simpel en zeker omdat Logisland gebruik maakt van docker-compose.

```

yum install docker

systemctl start docker

systemctl enable docker

sudo curl -L "https://github.com/docker/compose/releases/download/1.24.0/docker-compose-$(uname -s)-$(uname -m)" -o /usr/local/bin/docker-compose

sudo chmod +x /usr/local/bin/docker-compose

docker-compose -version

```

Elasticsearch heeft echter een grotere minimum configuratie van de map-count nodig dan hoe deze standaard ingesteld is. Dt moest ik dus eerst aanpassen.

```

sudo sysctl -w vm.max_map_count=262144

sudo sh -c 'echo "vm.max_map_count=262144" >> /etc/sysctl.conf

```

Nu dat de containers waren opgestart kon ik een aantal van de tutorials uitproberen. De docker-images komen standaard al met de configuratiefiles van de tutorial, dus was dit redelijk simpel om te testen.

```

sudo docker exec -i -t conf_logisland_1 bin/logisland.sh --conf conf/logs-to-events.yml

```

Dit commando zegt tegen docker “voer voor mij het commando in de bin-folder logisland.sh met de configuratie logs-to-events.yml in de conf folder uit op de

---

1https://github.com/Hurence/logisland

2https://logisland.readthedocs.io/en/latest/index.html

3ELK staat voor Elasticsearch, Logstash en Kibana een stack voor data-verwerking ontwikkelt door Apache.

conf_logisland_1 container". Waardoor de geselecteerde configuratie wordt uitgevoerd en de Logisland pipeline klaar staat om data te ontvangen.

3.5 Rebootstrappen van Lisbon.hypprod.inuits.eu

Na Logisland uitgeprobeerd te hebben op een lokale omgeving werd al snel duidelijk dat de beschikbare RAM, in mijn virtualbox, onvoldoende ging zijn. Kris en ik besliste om over te stappen op een van de Inuits' servers. Deze waren momenteel druk in de weer om ge-rebootstrapt [4]te worden van Centos 6 naar Centos 7. Dit was een fantastische uitdaging in mijn stage omdat dit een goede manier was om in aanraking te komen met zowel de dns, server hosting platformen (vb. OVH) en Puppet.

3.5.1 Vereisten

Deze onderdelen moeten geïnstalleerd zijn op het lokale werkstation om deze configuratie te kunnen uitvoeren:

- git
- Ansible
- vim

3.5.2 DNS instellen

Stap één is de DNS instellen om toekomstige problemen te vermijden en onze adressen al te reserveren. Hiervoor moeten we de versionned configuratie bestanden downloaden van de Redmine git repository.

` git clone ssh://git@redmine.inuits.eu:2223/inuits/inuits-infra/dns.git `

In de gedownloade repository vinden we een script om vrije ip-addressen weer te geven.

```

cd dns

./list_free_IP_adresses_in_OVH.sh

```

Na een vrij ip-adres te hebben gekozen moeten we dit ip address toevoegen aan de configuratie bestanden in dezelfde folder.

```

cd var/named/data

vim named.hypprod.inuits.eu

```

```

4(Re)-bootstrappen is de handeling van het opzetten van een service op een systeem zonder externe input. In dit geval het automatisch opzetten van de Puppet service waardoor installaties geautomatiseerd worden.
```

```
    ; lisbon vms
    lisbon           IN  A     10.0.64.29
```

Verander de Serial datum aan de bovenkant van het bestand.

```
    2014053000         ; Serial
```

Voeg logisland toe aan het named.playground.inuits.eu

` vim named.hypprod.inuits.eu `

```
    ; logisland vm
    logisland           IN  A     10.0.229.220
```

Verander de Serial datum aan de bovenkant van het bestand.

```
    2014053000         ; Serial
```

Nu moet nog de reverse DNS toegevoegt worden.

` vim 10.0.db `

```
    ; lisbon hosts
    29.64   IN  PTR lisbon.hypprod.inuits.eu.

    ; logisland host

    220.229 IN  PTR logisland.playground.inuits.eu.
```

Wanneer alle aanpassingen zijn gebeurt moeten we de wijzigingen terug pushen naar de git repository.

```
```

git add named.hypprod.inuits.eu 10.0.db

git commit -m “Added lisbon.hypprod”

git push

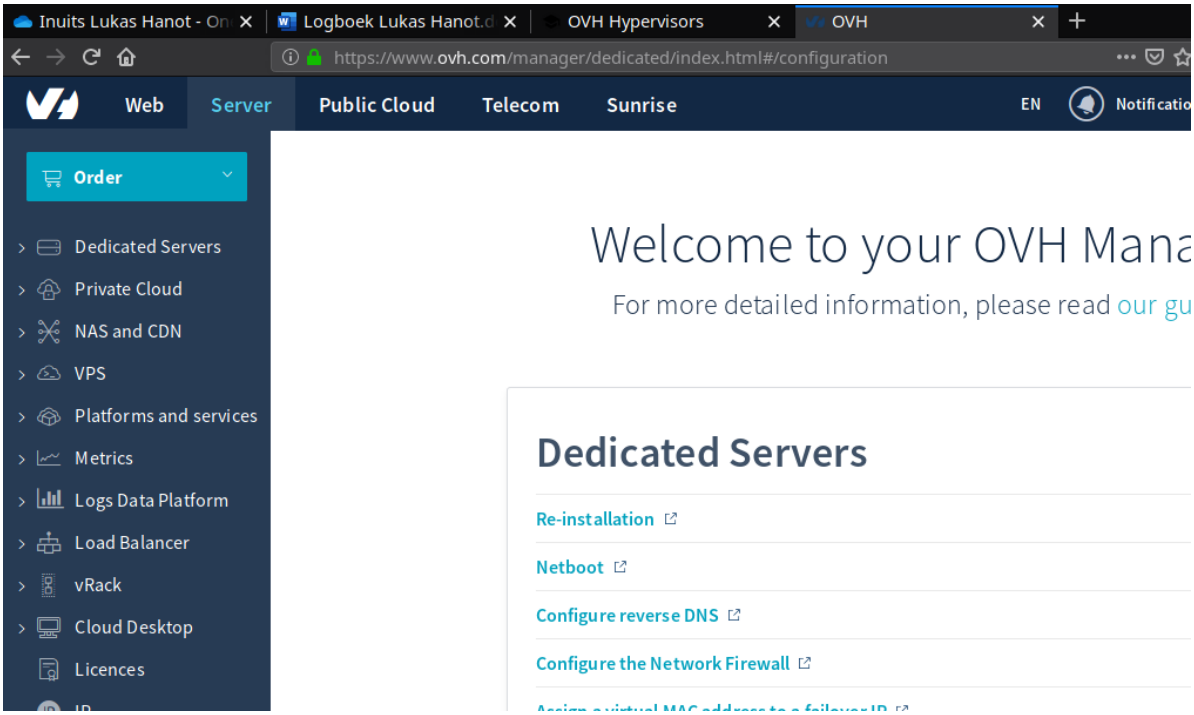
```

De wijzigingen worden nu automatisch uitgevoerd op de Inuits-infrastructuur.

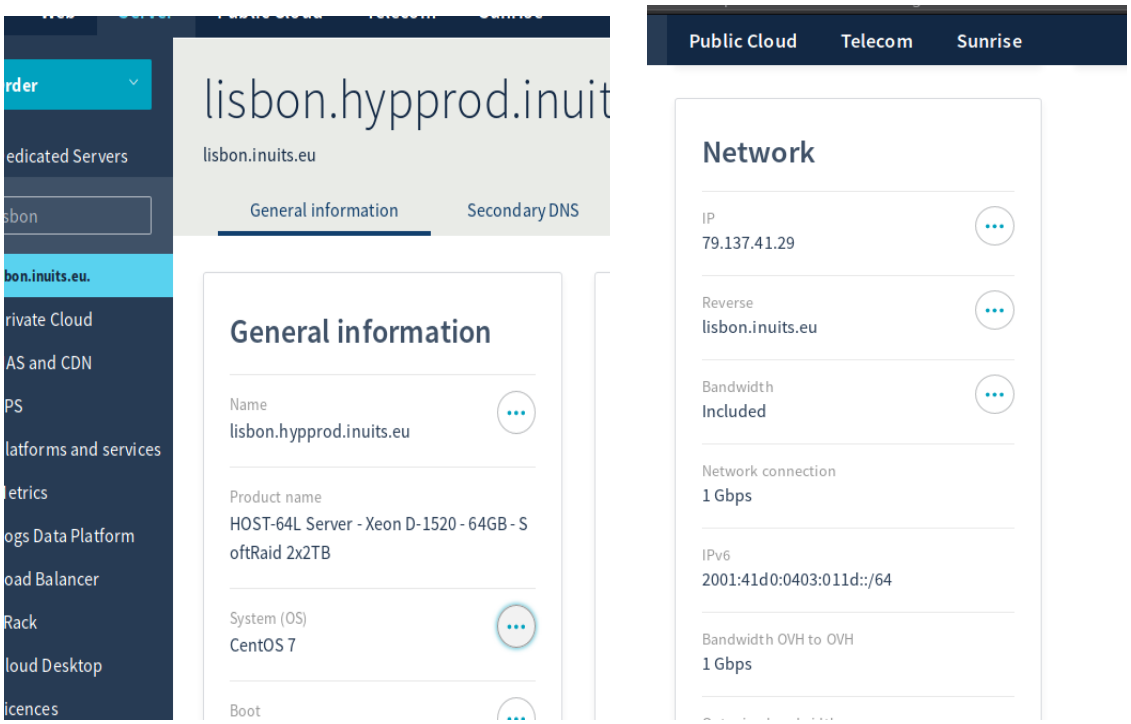
### 3.5.3 Centos upgraden

Navigeer naar de OVH login pagina https://www.ovh.com/auth/ en log hier in met de Inuits inloggegevens.

Klik in de rechterbovenhoek op ‘Control Panel’ en zoek dan onder server naar de te rebootstrappen server.


```

Wanneer u de server gevonden heeft, kan u de naam en netwerkgegevens wijzigen.



Nu gaan we de host herinstalleren, klik op de settings knop naast “System (OS)”. In het scherm dat nu verschijnt kies je voor ‘OVH template’ gevolgd door het gewenste besturingssysteem in dit geval ‘Centos 7’.

De-activeer de OVH monitoring.

Wanneer de installatie voltooid is, staat er een volledig nieuw proper besturingssysteem op de remote-host. Nu moeten we het systeem voorbereiden op Puppet waarna alle software door automatisatie kan worden geïnstalleerd.

Kloon de repo voor de Puppet infrastructuur

`ssh://git@redmine.inuits.eu:2223/inuits/inuits-infra/mgmt-environment/puppet-tree-mgmt.git`

Maak een nieuw puppet manifest aan waarin de Bootstrap beschreven staat.

```

vim mgmt-puppet5/manifests/nodes/lisbon.pp

```
node /london.hypprod.inuits.eu/ {

  include roles::hypervisor

  # install CentOS 7 by default
  Virtinstall::Bootstrap{
    ensure   => present,
    vm_ram   => '1280',   # bootstrap of CentOS 7 fails on
1024MB RAM!!!
    vm_os_version   => '7',
    vm_os_variant   => 'rhel7',
    vm_lvm_vg_name   => 'vg_london',
    bootstrap_url_mirror =>
'http://pulp.mgmtprod.inuits.eu/pulp/repos/pub/mirrors/centos/7/os/
x86_64/',
  }
}

```

### 3.5.4 puppet instellen voor rebootstrap

Nu we onze basis configuratie hebben aangemaakt is de volgende stap de host voorbereiden.

We halen het bootstrapping script van redmine:

`ssh://git@redmine.inuits.eu:2223/inuits/inuits-infra/hosted-bootstrap/hosted_bootstrap.git`

Start het provision.sh script dat we net hebben binnengehaald.

```
./provision.sh --host $<name of the machine> --dc ovh --public
$<Public_ip> --private $<private_ip>/22 --gateway $<vpngateway_ip> --
domain $<long_domain_name> --installdrive1 sda --installdrive2 sdb -r

example

./provision.sh --host lisbon --dc ovh --public 79.137.41.30 --private
10.0.64.29/22 --gateway 10.0.228.120 --domain hypprod.inuits.eu --
installdrive1 sda --installdrive2 sdb -r


```

Note: de inloggegevens die OVH heeft gemaild, moeten worden ingevoerd tijdens het runnen van dit script.

Dit installeert Centos, onze eerder gegenereerde configuratie op de host, waardoor we een Centos 7 systeem hebben, dat geïnstalleerd is in raid 1 met de nodige puppet
```

configuraties. Hierna kunnen we door de puppet configuratie aan te passen, virtuele machines op deze Centos installeren.

3.5.5 De server voorbereiden en connecteren aan puppet

Voeg de configuratie van de virtuele machine toe aan het lisbon.pp manifest

` vim ./mgmt-puppet5/manifests/nodes/lisbon.pp `

```
 virtinstall::bootstrap{'logisland.playground.inuits.eu':
   ensure               => present,
   vm_ipaddress                => '10.0.229.220',
   vm_vcpus             => '2',
   vm_ram               => '25360',
   plannedenvironment    => 'mgmtplay',
   puppetversion    => 5,
   puppetmaster     => 'puppet5-ovh-01.mgmtprod.inuits.eu',
   extra_volumes    => [{
    name      => 'logisland_var',
    fs  => 'ext4',
    device     => 'vdb',
   },
   {
    name      => 'redis-data',
    fs  => 'ext4',
    device     => 'vdc',
   },
   ],
 }


 logical_volume {'logisland_var':
   ensure               => present,
```

```
  volume_group     => 'vg_lisbon',
  size             => '40G',
 }

 logical_volume {'redis-data':
  ensure           => present,
  volume_group     => 'vg_lisbon',
  size             => '100G',
 }
```

En creëer een basis puppet configuratie voor de logisland vm

```vim ./mgmt-puppet5/manifests/nodes/logisland.pp ```

```
node /logisland.playground.inuits.eu/ {

  include ::profile_base
}
```

3.5.6 Een laatste puppet run

Connecteer via ssh naar de remote host en run daar een puppet synchronisatie om de configuratie te vernieuwen en de vm te installeren.

```
ssh lisbon.hypprod.inuits.eu
sudo puppet agent -t
sudo reboot
```
```

3.6 Installatie van Logisland in Logisland.playground.inuits.eu

Deze installatie maakt gebruik van docker maar Logisland kan ook afzonderlijk geïnstalleerd worden.

3.6.1 Vereisten

- Docker

- Docker-compose

- SSH

(Mogelijk moet de “mmap count” nog vergroot worden op de host zie hier)

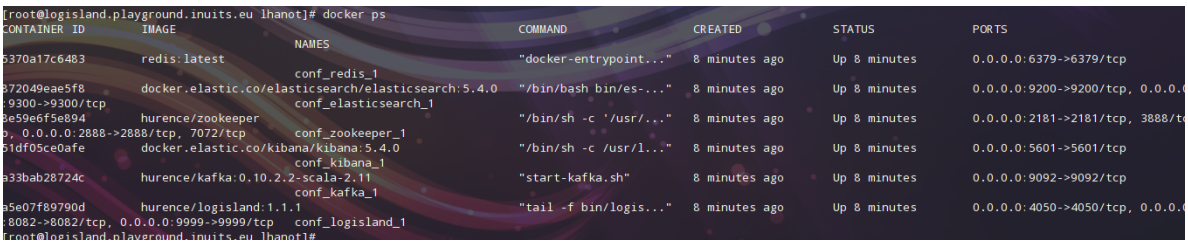
Docker-compose staat ons toe om een complexe container omgeving op te zetten aan de hand van een declaratief bestand. Dit bestand halen we op met dit commando:

```sh

curl -L https://raw.githubusercontent.com/Hurence/logisland/master/logisland-core/logisland-framework/logisland-:/src/main/:/conf/docker-compose.yml >> docker-compose.yml

```

3.6.2 Start docker

Om docker containers te kunnen gebruiken moet natuurlijk eerst de docker service opgestart worden. Met de volgende commando’s zorgen we eerst dat de service gestart wordt en automatisch opgestart wordt bij volgende heropstart. Waarna we de docker containers opstarten in een tmux omgeving hierdoor is deze shell bereikbaar vanuit andere remote connecties. Het laatste commando geeft ons alle actieve containers terug waarmee we controleren of de opstart geslaagd is.

```sh

sudo systemctl enable --now docker

tmux

sudo docker-compose up

ctrl-b + d

docker ps #This shows all running containers

```



3.6.3 Configuratie van Logisland

Logisland komt met een aantal configuraties ingebakken: deze kunnen terug gevonden worden in de Logisland container. Om deze eruit te halen run dit commando:

`docker cp conf_logisland_1:/opt/logisland/conf/* .`

Dit is de index-apache-logs-es.yml configuratie:

```yml
#########################################################################################

## Logisland configuration script tempate

#########################################################################################


version: 1.1.1

documentation: LogIsland analytics main config file. Put here every engine or component config


#########################################################################################

## engine

engine:

  component: com.hurence.logisland.engine.spark.KafkaStreamProcessingEngine

  type: engine

  documentation: Index some apache logs with logisland

  configuration:

    spark.app.name: IndexApacheLogsDemo
```

```
    spark.master: local[2]
    spark.driver.memory: 1G
    spark.driver.cores: 1
    spark.executor.memory: 2G
    spark.executor.instances: 4
    spark.executor.cores: 2
    spark.yarn.queue: default
    spark.yarn.maxAppAttempts: 4
    spark.yarn.am.attemptFailuresValidityInterval: 1h
    spark.yarn.max.executor.failures: 20
    spark.yarn.executor.failuresValidityInterval: 1h
    spark.task.maxFailures: 8
    spark.serializer: org.apache.spark.serializer.KryoSerializer
    spark.streaming.batchDuration: 1000
    spark.streaming.backpressure.enabled: false
    spark.streaming.unpersist: false
    spark.streaming.blockInterval: 500
    spark.streaming.kafka.maxRatePerPartition: 3000
    spark.streaming.timeout: -1
    spark.streaming.kafka.maxRetries: 3
    spark.streaming.ui.retainedBatches: 200
    spark.streaming.receiver.writeAheadLog.enable: false
    spark.ui.port: 4050

  controllerServiceConfigurations:

    - controllerService: elasticsearch_service
      component:
com.hurence.logisland.service.elasticsearch.Elasticsearch_5_4_0_ClientService
      type: service
```

```
    documentation: elasticsearch service
    configuration:
      hosts: ${ES_HOSTS}
      cluster.name: ${ES_CLUSTER_NAME}
      batch.size: 5000


  streamConfigurations:


    # main processing stream
    - stream: parsing_stream
      component:
com.hurence.logisland.stream.spark.KafkaRecordStreamParallelProcessing
      type: stream
      documentation: a processor that converts raw apache logs into structured log records
      configuration:
        kafka.input.topics: logisland_raw
        kafka.output.topics: logisland_events
        kafka.error.topics: logisland_errors
        kafka.input.topics.serializer: none
        kafka.output.topics.serializer: com.hurence.logisland.serializer.KryoSerializer
        kafka.error.topics.serializer: com.hurence.logisland.serializer.JsonSerializer
        kafka.metadata.broker.list: ${KAFKA_BROKERS}
        kafka.zookeeper.quorum: ${ZK_QUORUM}
        kafka.topic.autoCreate: true
        kafka.topic.default.partitions: 4
        kafka.topic.default.replicationFactor: 1
      processorConfigurations:


        # parse apache logs into logisland records
        - processor: apache_parser
```

```
        component: com.hurence.logisland.processor.SplitText

        type: parser

        documentation: a parser that produce events from an apache log REGEX

        configuration:

          record.type: apache_log

          value.regex: (\S+)\s+(\S+)\s+(\S+)\s+\[([\w:\/]+\s[+\-]\d{4})\]\s+"(\S+)\s+(\S+)\s*(\
S*)"\s+(\S+)\s+(\S+)

          value.fields:
src_ip,identd,user,record_time,http_method,http_query,http_version,http_status,bytes_out



       # all the parsed records are added to elasticsearch by bulk

       - processor: es_publisher

         component: com.hurence.logisland.processor.elasticsearch.BulkAddElasticsearch

         type: processor

         documentation: a processor that indexes processed events in elasticsearch

         configuration:

           elasticsearch.client.service: elasticsearch_service

           default.index: logisland

           default.type: event

           timebased.index: yesterday

           es.index.field: search_index

           es.type.field: record_type
```

De configuratie hierboven configureert Kafka, Spark and Elasticsearch. De listener op de Kafka host geïnitialiseerd als logisland_raw topic zodat deze input kan accepteren en deze opslaan als ruwe log data. Spark is geconfigureerd om deze ruwe data af te halen van logisland_raw, de data te classeren met behulp van een regex en erna alle output door te sturen naar de Elasticsearch publisher.

In vergelijking met het voorgaande voorbeeld vindt u hieronder de actuele configuratie:

```yml
```

```
#########################################################################
########################################
```

Logisland configuration script template

```
#########################################################################
########################################
```

version: 1.1.1

documentation: LogIsland analytics main config file. Put here every engine or component config

```
#########################################################################
########################################
```

engine

```
engine:
  component: com.hurence.logisland.engine.spark.KafkaStreamProcessingEngine
  type: engine
  documentation: A logisland stream that match custom queries
  configuration:
    spark.app.name: InuitsLogsToEvents
    spark.master: local[4]
    spark.driver.memory: 10g
    spark.driver.cores: 4
    spark.executor.memory: 10g
    spark.executor.instances: 8
    spark.executor.cores: 6
    spark.yarn.queue: default
    spark.yarn.maxAppAttempts: 4
    spark.yarn.am.attemptFailuresValidityInterval: 1h
    spark.yarn.max.executor.failures: 20
    spark.yarn.executor.failuresValidityInterval: 1h
    spark.task.maxFailures: 8
```

```
    spark.serializer: org.apache.spark.serializer.KryoSerializer
    spark.streaming.batchDuration: 2000
    spark.streaming.backpressure.enabled: false
    spark.streaming.blockInterval: 100
    spark.streaming.kafka.maxRatePerPartition: 4000
    spark.streaming.timeout: -1
    spark.streaming.unpersist: false
    spark.streaming.kafka.maxRetries: 3
    spark.streaming.ui.retainedBatches: 200
    spark.streaming.receiver.writeAheadLog.enable: false
    spark.ui.port: 4050


  controllerServiceConfigurations:


    - controllerService: elasticsearch_service
      component:
com.hurence.logisland.service.elasticsearch.Elasticsearch_5_4_0_ClientService
      type: service
      documentation: elasticsearch service
      configuration:
        hosts: ${ES_HOSTS}
        cluster.name: ${ES_CLUSTER_NAME}
        batch.size: 1000


    - controllerService: datastore_service
      component: com.hurence.logisland.redis.service.RedisKeyValueCacheService
      type: service
      documentation: redis datastore service
      configuration:
        connection.string: redis:6379
```

```
      redis.mode: standalone
      database.index: 0
      communication.timeout: 10 seconds
      pool.max.total: 8
      pool.max.idle: 8
      pool.min.idle: 0
      pool.block.when.exhausted: true
      pool.max.wait.time: 10 seconds
      pool.min.evictable.idle.time: 60 seconds
      pool.time.between.eviction.runs: 30 seconds
      pool.num.tests.per.eviction.run: -1
      pool.test.on.create: false
      pool.test.on.borrow: false
      pool.test.on.return: false
      pool.test.while.idle: true
      record.recordSerializer: com.hurence.logisland.serializer.JsonSerializer


  streamConfigurations:


    # structure logs
    - stream: parsing_stream
      component:
com.hurence.logisland.stream.spark.KafkaRecordStreamParallelProcessing
      type: stream
      documentation: a processor that converts raw logs into structured log records
      configuration:
        kafka.input.topics: logisland_raw
        kafka.output.topics: logisland_events
        kafka.error.topics: logisland_errors
        kafka.input.topics.serializer: none
```

```
      kafka.output.topics.serializer: com.hurence.logisland.serializer.JsonSerializer
      kafka.error.topics.serializer: com.hurence.logisland.serializer.JsonSerializer
      kafka.metadata.broker.list: ${KAFKA_BROKERS}
      kafka.zookeeper.quorum: ${ZK_QUORUM}
      kafka.topic.autoCreate: true
      kafka.topic.default.partitions: 4
      kafka.topic.default.replicationFactor: 1
    processorConfigurations:
      - processor: log_parser
        component: com.hurence.logisland.processor.SplitText
        type: parser
        documentation: Gets individual fields out of log string
        configuration:
          record.type: kv_record


          value.regex: (\S+)\s+(\S+)\s+<181>(\w+\ \d{1,2}\ \d{2}\:\d{2}\:\d{2})\s+(\S+)\s+(\S+)\
s+\S+\=(\S+)\s+\S+\=(\S+)\s+\S+\=(\S+)\s+\S+\=(\S+)\s+\S+\=(\S+)\s+\S+\=(\S+)\s+\S+\=(\
S+)\s+(msg='.*')

          value.fields: timestamp,host,datum,logsource,program,type_log,message_audit,
pid, uid, auid, ses, subj, message


##       - processor: field_types_converter
##         component: com.hurence.logisland.processor.ConvertFieldsType
##         type: processor
##         documentation: convert some field to a given type
##         configuration:
##           bytes_out: long
```

```
    # indexing stream
    - stream: indexing_stream
      component:
com.hurence.logisland.stream.spark.KafkaRecordStreamParallelProcessing
      type: stream
      documentation: a processor that converts raw logs into structured log records
      configuration:
        kafka.input.topics: logisland_events,logisland_alerts,logisland_aggregations
        kafka.output.topics: none
        kafka.error.topics: logisland_errors
        kafka.input.topics.serializer: com.hurence.logisland.serializer.JsonSerializer
        kafka.output.topics.serializer: com.hurence.logisland.serializer.JsonSerializer
        kafka.error.topics.serializer: com.hurence.logisland.serializer.JsonSerializer
        kafka.metadata.broker.list: ${KAFKA_BROKERS}
        kafka.zookeeper.quorum: ${ZK_QUORUM}
        kafka.topic.autoCreate: true
        kafka.topic.default.partitions: 6
        kafka.topic.default.replicationFactor: 1
      processorConfigurations:
        - processor: es_publisher
          component: com.hurence.logisland.processor.elasticsearch.BulkAddElasticsearch
          type: processor
          documentation: a processor that indexes processed events in elasticsearch
          configuration:
            elasticsearch.client.service: elasticsearch_service
            default.index: logisland
            default.type: event
            timebased.index: yesterday
            es.index.field: search_index
            es.type.field: record_type
```

```
    # all the parsed records are added to datastore by bulk
    - processor: datastore_publisher
      component: com.hurence.logisland.processor.datastore.BulkPut
      type: processor
      documentation: "indexes processed events in datastore"
      configuration:
        datastore.client.service: datastore_service
```

In deze configuratie vinden we een vergroting van de minimum resources. Deze settings zijn aangepast naar de verwachte noden van onze testen. De data-parser is aangepast om onze toekomstige data om te vormen naar records in plaats van de originele apache-logs. Een redis output is toegevoegd om ook een cold storage te voorzien voor data waar zwaardere berekeningen op mogelijk zijn.

Deze configuratie moeten we toevoegen aan onze container.

```sh
docker cp ./inuits-logs-to-events.yml conf_logisland_1:/opt/logisland/conf/
```

Wanneer de configuratie gebeurd is, kunnen we Logisland opstarten.

```sh
tmux
docker exec -i -t conf_logisland_1 bin/logisland.sh --conf conf/inuits-logs-to-events.yml
Ctrl-b + d
```

3.6.4 Logstash

Logstash is de source waar alle data vandaan zal komen in deze POC. Logisland accepteert nog andere bronnen maar deze was al voorhanden binnen Inuits en leunt aan bij de vereisten van de POC.

Logstash wordt op de elastic.co site beschreven als:

“Logstash is an open source, server-side data processing pipeline that ingests data from a multitude of sources simultaneously, transforms it, and then sends it to your favorite “stash.” “

In deze POC wordt Logstash gebruikt om logs te verzamelen, te filteren en dan door te geven aan Kafka.

Om de connectie te leggen met Kafka moeten we één setting toevoegen aan de Logstash applicatie. Hieronder connecteren we naar de Logstash host en voeren we de nieuwe output toe aan de configuratie.

`ssh logstash01.secmgmt.inuits.eu`

change logstash settings

`cd /etc/logstash/conf.d/`

add to output:

```
output {
  kafka {
    bootstrap_servers => "logisland.playground.inuits.eu:9092"
    topic_id => 'logisland_raw'
  }
}
```

herstart Logstash om de configuratie te activeren.

`sudo systemctl restart logstash`

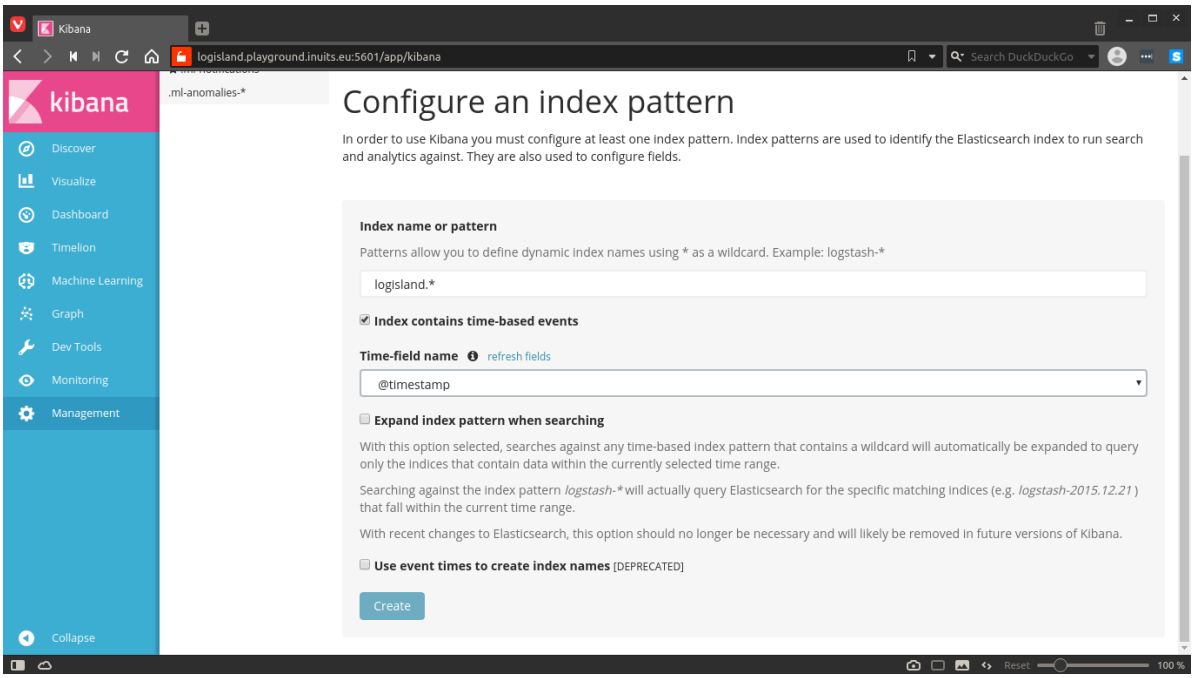
3.6.5 Link Kibana

Kibana visualiseert de data opgeslagen in Elasticsearch, waardoor we de data kunnen bestuderen en vergelijken.

Enkel de index van de Elasticsearch kluster moet toegevoegd worden om Kibana te doen werken. Dus ga naar:

` http://logisland.playground.inuits.eu:5601/app/kibana#/management/kibana/index?_g=() `

En hier wijzigen we “logstash-*” naar “logisland.*” en onder “Time-field name” selecteren we ‘timestamp’ waarna we op “Create” klikken.



3.7 Hoe werkt Logisland?

Nu hebben we de volledige configuratie van Logisland gedaan maar weten we nog niet hoe Logisland werkt. Dit hoofstuk hoopt hierin meer duidelijkheid te scheppen. We zullen stap voor stap door de Logisland configuratie gaan om parameters van aandacht aan te duiden en te verduidelijken.

3.7.1 The Engine

Dit is de basis Spark configuratie: de naam van de app, hoeveel geheugen gealloceerd moet worden, hoe de queue wordt georganiseerd en de data gestreamd moet worden. Streaming betekent hier hoe 1 ketting van records moet worden afgehaald van Kafka, verwerkt wordt en dan wordt teruggestuurd.

```yml

## engine

engine:

  component: com.hurence.logisland.engine.spark.KafkaStreamProcessingEngine

  type: engine

  documentation: A logisland stream that match custom queries

  configuration:

    spark.app.name: InuitsLogsToEvents

    spark.master: local[4]

    spark.driver.memory: 10g

    spark.driver.cores: 4

    spark.executor.memory: 10g

    spark.executor.instances: 8

    spark.executor.cores: 6

    spark.yarn.queue: default

    spark.yarn.maxAppAttempts: 4

    spark.yarn.am.attemptFailuresValidityInterval: 1h

    spark.yarn.max.executor.failures: 20

    spark.yarn.executor.failuresValidityInterval: 1h

    spark.task.maxFailures: 8

    spark.serializer: org.apache.spark.serializer.KryoSerializer
```

spark.streaming.batchDuration: 2000

spark.streaming.backpressure.enabled: false

spark.streaming.blockInterval: 100

spark.streaming.kafka.maxRatePerPartition: 4000

spark.streaming.timeout: -1

spark.streaming.unpersist: false

spark.streaming.kafka.maxRetries: 3

spark.streaming.ui.retainedBatches: 200

spark.streaming.receiver.writeAheadLog.enable: false

spark.ui.port: 4050

```

### 3.7.2 The Controller

De controllers zijn de output-services, die verbonden zijn aan de Spark-engine. In het voorbeeld hieronder kunnen we zien hoe zowel een Elasticsearch- controller als een Redis-controller wordt aangemaakt.

```yml

controllerServiceConfigurations:

- controllerService: elasticsearch_service

component:
com.hurence.logisland.service.elasticsearch.Elasticsearch_5_4_0_ClientService

type: service

documentation: elasticsearch service

configuration:

hosts: ${ES_HOSTS}

cluster.name: ${ES_CLUSTER_NAME}

batch.size: 1000

- controllerService: datastore_service

component: com.hurence.logisland.redis.service.RedisKeyValueCacheService

```
    type: service
    documentation: redis datastore service
    configuration:
      connection.string: redis:6379
      redis.mode: standalone
      database.index: 0
      communication.timeout: 10 seconds
      pool.max.total: 8
      pool.max.idle: 8
      pool.min.idle: 0
      pool.block.when.exhausted: true
      pool.max.wait.time: 10 seconds
      pool.min.evictable.idle.time: 60 seconds
      pool.time.between.eviction.runs: 30 seconds
      pool.num.tests.per.eviction.run: -1
      pool.test.on.create: false
      pool.test.on.borrow: false
      pool.test.on.return: false
      pool.test.while.idle: true
      record.recordSerializer: com.hurence.logisland.serializer.JsonSerializer
```

3.7.3 The Streams

De StreamConfiguration zijn de hersenen van Logisland. Hier wordt de data gelezen, bewerkt en uiteindelijk teruggezonden naar Kafka.

In deze configuratie duid het component aan wat er met de data zal gebeuren. De input en output parameters geven aan waar de data vandaan komt en hoe deze wordt weggeschreven. In dit voorbeeld verzamelen we data van de input topic 'logisland_raw' en sturen we data terug naar de output topic 'logisland_events' of de error topic 'logisland_errors'. Serializers verklaren aan Kafka hoe de data gevormd zal zijn bij het binnenkomen en hoe we het willen vormen bij het verzenden.

Dit wordt gevolgd door de processor-configuraties: hier gaan we aangeven hoe onze data moet worden omgevormd van de ruwe logs naar onze nieuwe data-patterns. We beginnen met de SplitText component deze wordt gebruikt om onze lange logstrings op te delen in afzonderlijke fields met behulp van een regex. Als extra staat er in de commented sectie een ConvertFieldsType processor deze probeert een field-type aan te passen naar een andere gewenste dataset.

```yml
streamConfigurations:

  # structure logs
  - stream: parsing_stream
    component: com.hurence.logisland.stream.spark.KafkaRecordStreamParallelProcessing
    type: stream
    documentation: a processor that converts raw logs into structured log records
    configuration:
      kafka.input.topics: logisland_raw
      kafka.output.topics: logisland_events
      kafka.error.topics: logisland_errors
      kafka.input.topics.serializer: none
      kafka.output.topics.serializer: com.hurence.logisland.serializer.JsonSerializer
      kafka.error.topics.serializer: com.hurence.logisland.serializer.JsonSerializer
      kafka.metadata.broker.list: ${KAFKA_BROKERS}
```

```
      kafka.zookeeper.quorum: ${ZK_QUORUM}

      kafka.topic.autoCreate: true

      kafka.topic.default.partitions: 4

      kafka.topic.default.replicationFactor: 1

    processorConfigurations:

      - processor: log_parser

        component: com.hurence.logisland.processor.SplitText

        type: parser

        documentation: Gets individual fields out of log string

        configuration:

          record.type: kv_record

          value.regex: (\S+)\s+(\S+)\s+<181>(\w+\ \d{1,2}\ \d{2}\:\d{2}\:\d{2})\s+(\S+)\s+(\S+)\
s+\S+\=(\S+)\s+\S+\=(\S+)\s+\S+\=(\S+)\s+\S+\=(\S+)\s+\S+\=(\S+)\s+\S+\=(\S+)\s+\S+\=(\
S+)\s+(msg='.*')

          value.fields: timestamp,host,datum,logsource,program,type_log,message_audit,
pid, uid, auid, ses, subj, message


##       - processor: field_types_converter

##         component: com.hurence.logisland.processor.ConvertFieldsType

##         type: processor

##         documentation: convert some field to a given type

##         configuration:

##           bytes_out: long
```

Voor meer voorbeelden kijk hier.

3.7.4 The Indexer

De Indexer is verantwoordelijk voor de data-output van de Logisland-applicatie. Deze output kan allerhande output sources, die eerder gedefineerd werden in de controller, leiden. Het werkt op dezelfde manier als de eerdere streams namelijk door de connectie naar Kafka, de input, te definiëren. Alle data wordt echter een laatste keer door Spark gehaald waardoor formatering mogelijk is en daarna volledig kan worden weggeschreven naar Elasticsearch of Redis.

```yml
   # indexing stream
   - stream: indexing_stream
     component:
com.hurence.logisland.stream.spark.KafkaRecordStreamParallelProcessing
     type: stream
     documentation: a processor that converts raw logs into structured log records
     configuration:
       kafka.input.topics: logisland_events,logisland_alerts,logisland_aggregations
       kafka.output.topics: none
       kafka.error.topics: logisland_errors
       kafka.input.topics.serializer: com.hurence.logisland.serializer.JsonSerializer
       kafka.output.topics.serializer: com.hurence.logisland.serializer.JsonSerializer
       kafka.error.topics.serializer: com.hurence.logisland.serializer.JsonSerializer
       kafka.metadata.broker.list: ${KAFKA_BROKERS}
       kafka.zookeeper.quorum: ${ZK_QUORUM}
       kafka.topic.autoCreate: true
       kafka.topic.default.partitions: 6
       kafka.topic.default.replicationFactor: 1
     processorConfigurations:
       - processor: es_publisher
         component: com.hurence.logisland.processor.elasticsearch.BulkAddElasticsearch
         type: processor
         documentation: a processor that indexes processed events in elasticsearch
```

```
    configuration:
      elasticsearch.client.service: elasticsearch_service
      default.index: logisland
      default.type: event
      timebased.index: yesterday
      es.index.field: search_index
      es.type.field: record_type


  # all the parsed records are added to datastore by bulk
  - processor: datastore_publisher
    component: com.hurence.logisland.processor.datastore.BulkPut
    type: processor
    documentation: "indexes processed events in datastore"
    configuration:
      datastore.client.service: datastore_service
```

3.7.5 Managementcommands

Docker

Command	Function
docker ps	Show active docker containers
docker exec	Run a command inside a container (also usefull for creating a shell inside a container)
docker cp	Copy-paste a file to or from a container
docker (re)start/stop	Start, stop or restart a container
docker system prune	Clean-up all stopped containers, images and volumes

Elasticsearch

Deze commandos kunnen ingevoerd worden zowel in de "kibana dev tools" als via curl -X commandos

command	function
GET 'http://localhost:9200/\<index name>'	Returns an index
POST 'http://localhost:9200/\<index name>'	Sends data to Elasticsearch
DELETE 'http://localhost:9200/\<index name>'	Delete an item on Elasticsearch

Send data to Kafka without Logstash

Om data input te testen kan simpelweg data via een Kafka script verzonden worden naar de Kafka-listener. Hierdoor is een data-source zoals logstash overbodig.

```sh

# ${KAFKA_HOME} is de locatie waar Kafka geïnstalleerd is

STDOUT | ${KAFKA_HOME}/bin/kafka-console-producer.sh --broker-list kafka:9092 --topic logisland_raw`

```

3.7.6 Probleem oplossen

Elasticsearch extend mmap

Elasticsearch gebruikt een hybrid mmapfs om standaard opslagindices te bepalen. Deze instellingen zijn op een standaard besturingssysteem meestal ontoereikend voor de doeleinden van Logisland. Deze limiet kan met het volgende commando worden uitgebreid.

```sh

sudo sysctl -w vm.max_map_count=262144

sudo sh -c 'echo "vm.max_map_count=262144" >> /etc/sysctl.conf'

```

Kafka connection error

Connecteer naar de Kafka container.

`sudo docker exec -i -t conf_kafka_1 bash`

Wijzig de Kafka instellingen.

`vim /opt/kafka.versionnumber/config/server.properties`

```yml

listeners = PLAINTEXT://0.0.0.0:9092

advertised.listeners = PLAINTEXT://kafka:9092

```

Kafka stelt automatisch de ingegeven advertiser in als leader dus als dit een extern ip is geraakt Kafka in de war. Dit lossen we op door de listener in te stellen als “kafka” en “kafka” toe te voegen aan de container /etc/hosts file. Om externe servers te laten connecteren met Kafka moeten we ook op de hypervisor-Kafka instellen in de /etc/hosts file

Diskspace warnings

Omdat Logisland veel data verzamelt op korte tijd, was het in mijn test- omgeving soms nodig om terug ruimte vrij te maken voor verdere experimenten. Dankzij de container-technologie is het echter enorm simpel om de containers te verwijderen en terug proper op te starten.

```

$ docker-compose -f /opt/logisland/conf/docker-compose.yml down

$ docker system prune
```

$ docker-compose - f /opt/logisland/conf/docker-compose.yml up

$ docker exec -i -t conf_kafka_1 bash

```

Change settings in kafka

```

vim /opt/kafka/config/server.properties

Set these in file

listeners=PLAINTEXT://0.0.0.0:9092

advertised.listeners=PLAINTEXT://kafka:9092

exit

$ docker restart conf_kafka_1

```

```

docker exec -i -t conf_logisland_1 bin/logisland.sh --conf conf/inuits-logs-to-events.yml

```
```

3.7.7 Spark stream processing examples

Dankzij de korte uitleg hierboven en deze te koppen aan de hieronder uitgeschreven voorbeelden zou het niet moeilijk moeten zijn om zelf een config uit te werken. De uitleg kan teruggevonden worden in het Hoe werkt Logisland hoofdstuk.

Time sampling

```yml
## parsing time series
- stream: parsing_stream
  component: com.hurence.logisland.stream.spark.KafkaRecordStreamParallelProcessing
  type: stream
  documentation: a processor that links
  configuration:
    kafka.input.topics: logisland_ts_raw
    kafka.output.topics: logisland_ts_events
    kafka.error.topics: logisland_errors
    kafka.input.topics.serializer: none
    kafka.output.topics.serializer: com.hurence.logisland.serializer.KryoSerializer
    kafka.error.topics.serializer: com.hurence.logisland.serializer.JsonSerializer
    avro.output.schema: >
     {  "version":1,
        "type": "record",
        "name": "com.hurence.logisland.record.cpu_usage",
        "fields": [
          { "name": "record_errors",   "type": [ {"type": "array", "items": "string"},"null"] },
          { "name": "record_raw_key", "type": ["string","null"] },
          { "name": "record_raw_value", "type": ["string","null"] },
          { "name": "record_id",   "type": ["string"] },
          { "name": "record_time", "type": ["long"] },
          { "name": "record_type", "type": ["string"] },
          { "name": "record_value",      "type": ["string","null"] } ]}
```

```
    kafka.metadata.broker.list: sandbox:9092
    kafka.zookeeper.quorum: sandbox:2181
    kafka.topic.autoCreate: true
    kafka.topic.default.partitions: 4
    kafka.topic.default.replicationFactor: 1
  processorConfigurations:
    - processor: apache_parser
      component: com.hurence.logisland.processor.SplitText
      type: parser
      documentation: a parser that produce events from an apache log REGEX
      configuration:
        record.type: apache_log
        value.regex: (\S+),(\S+)
        value.fields: record_time,record_value

- stream: detect_outliers
  component: com.hurence.logisland.stream.spark.KafkaRecordStreamParallelProcessing
  type: stream
  documentation: a processor that match query in parrallel
  configuration:
    kafka.input.topics: logisland_sensor_events
    kafka.output.topics: logisland_sensor_outliers_events
    kafka.error.topics: logisland_errors
    kafka.input.topics.serializer: com.hurence.logisland.serializer.KryoSerializer
    kafka.output.topics.serializer: com.hurence.logisland.serializer.KryoSerializer
    kafka.error.topics.serializer: com.hurence.logisland.serializer.JsonSerializer
    kafka.metadata.broker.list: sandbox:9092
    kafka.zookeeper.quorum: sandbox:2181
    kafka.topic.autoCreate: true
```

```
    kafka.topic.default.partitions: 2
    kafka.topic.default.replicationFactor: 1
  processorConfigurations:
    - processor: match_query
      component: com.hurence.logisland.processor.DetectOutliers
      type: processor
      documentation: a processor that detection something exotic in a continuous time
series values
      configuration:
        rotation.policy.type: by_amount
        rotation.policy.amount: 100
        rotation.policy.unit: points
        chunking.policy.type: by_amount
        chunking.policy.amount: 10
        chunking.policy.unit: points
        global.statistics.min: -100000
        min.amount.to.predict: 100
        zscore.cutoffs.normal: 3.5
        zscore.cutoffs.moderate: 5
        record.value.field: record_value
        record.time.field: record_time
        output.record.type: sensor_outlier
```

sample time series

```
- stream: detect_outliers
  component: com.hurence.logisland.stream.spark.KafkaRecordStreamParallelProcessing
  type: stream
  documentation: a processor that match query in parrallel
  configuration:
    kafka.input.topics: logisland_sensor_events
    kafka.output.topics: logisland_sensor_sampled_events
    kafka.error.topics: logisland_errors
    kafka.input.topics.serializer: com.hurence.logisland.serializer.KryoSerializer
    kafka.output.topics.serializer: com.hurence.logisland.serializer.KryoSerializer
    kafka.error.topics.serializer: com.hurence.logisland.serializer.JsonSerializer
    kafka.metadata.broker.list: sandbox:9092
    kafka.zookeeper.quorum: sandbox:2181
    kafka.topic.autoCreate: true
    kafka.topic.default.partitions: 2
    kafka.topic.default.replicationFactor: 1
  processorConfigurations:
    - processor: sampler
      component: com.hurence.logisland.processor.SampleRecords
      type: processor
      documentation: a processor that reduce the number of time series values
      configuration:
        record.value.field: record_value
        record.time.field: record_time
        sampling.algorithm: average
        sampling.parameter: 10

```

3.7.8 Processors

match queries

Deze processor kan gebruikt worden om in een stream enkel records terug te geven, die voldoen aan de vereisten.

```yml
- processor: match_query
  component: com.hurence.logisland.processor.MatchQuery
  type: processor
  documentation: a parser that matches lucene queries on records
  configuration:
    policy.onmiss: forward
    policy.onmatch: all
    blacklisted_host: src_ip:(+alyssa +prodigy)
    montana_host: src_ip:montana
```

Normalize field

```yml
- processor: normalize_fields
  component: com.hurence.logisland.processor.NormalizeFields
  type: parser
  documentation: change field name 'bytes_out' to `record_value`
  configuration:
    conflict.resolution.policy: overwrite_existing
    record_value: bytes_out
```

Modify id

```yml
- processor: modify_id
  component: com.hurence.logisland.processor.ModifyId
  type: parser
  documentation: change current id to src_ip
  configuration:
    id.generation.strategy: fromFields
    fields.to.hash: src_ip
    java.formatter.string: "%1$s"
```

Remove fields

```yml
- processor: remove_fields
  component: com.hurence.logisland.processor.RemoveFields
  type: parser
  documentation: remove useless fields
  configuration:
    fields.to.remove:
src_ip,identd,user,http_method,http_query,http_version,http_status,bytes_out
```

ConvertField

```yml
- processor: cast
  component: com.hurence.logisland.processor.ConvertFieldsType
  type: parser
  documentation: cast values
  configuration:
    record_value: double
```

```

**ComputeThresholds**

```yml
- processor: compute_thresholds
  component: com.hurence.logisland.processor.alerting.CheckThresholds
  type: processor
  documentation: |
    compute threshold cross from given formulas.
    each dynamic property will return a new record according to the formula definition
    the record name will be set to the property name
    the record time will be set to the current timestamp

    a threshold_cross has the following properties : count, time, duration, value
  configuration:
    datastore.client.service: datastore_service
    output.record.type: threshold_cross
    max.cpu.time: 100
    max.memory: 12800000
    max.prepared.statements: 5
    record.ttl: 300000
    threshold1: cache("computed1").value > 2000.0
```

ComputeAlerts

```yml
- processor: compute_alerts1
  component: com.hurence.logisland.processor.alerting.CheckAlerts
  type: processor
  documentation: |
    compute threshold cross from given formulas.
```
```

```
  each dynamic property will return a new record according to the formula definition
  the record name will be set to the property name
  the record time will be set to the current timestamp
 configuration:
  datastore.client.service: datastore_service
  output.record.type: medium_alert
  alert.criticity: 1
  max.cpu.time: 100
  max.memory: 12800000
  max.prepared.statements: 5
  profile.activation.condition: cache("threshold1").value > 3000.0
  alert1: cache("threshold1").duration > 50.0
```

4 CONCLUSIE

Wat een opdracht! Test de Logisland-applicatie en vind uit of ze bruikbaar zou kunnen zijn voor mijn stagebedrijf Inuits. Ook al mocht ik er 3 maanden aan werken, toch leek me in eerste instantie zoiets quasi onmogelijk. Maar kijk, 3 maanden later, ben ik er, weliswaar met veel hulp, in geslaagd om de applicatie te laten draaien op mijn eigen PC en nadien te embedden in de infrastructuur van Inuits. So far so good. Dus dacht ik laat die vakantie in Logisland maar beginnen...

Ik testte verschillende scenario's uit maar ik ontdekte vrij snel dat de Logisland-applicatie weinig meer bood dan de funcionaliteit, die ik al eerder tegen kwam bij het bekijken van hoe Logstash momenteel al werkt bij Inuits. En juist dat zou zogezegd één van de belangrijkste onderdelen van Logisland zijn. Deze functionaliteit was weliswaar pas recenter toegevoegd aan Logstash maar maakte een groot deel van de voordelen van Logisland plots minder interessant.

Initieel werd me gevraagd om de tool uit te proberen op mijn eigen PC om zo makkellijker kennis te maken met de tool an sich. Ik botste echter snel op een aantal limitaties: het geheugen van mijn laptop was onoereikend om de tool te draaien. Dan zat er niets anders op dan de tool op de Inuits-server te plaatsen. Tijdens het testen, ontdekte ik dat een eerste run op data perfect verliep maar bij het doorsturen van nieuwe data de tool errors gaf. De tool accepteert dus eigenlijk maar 1 soort data.

Ik stelde vast dat er weinig documentatie vanuit Hurence beschikbaar is en ook online is er weinig te vinden. Dat vergemakkelijkte mijn taak niet. Dit heeft vooral te maken met het feit dat ze zelf commerciële support en opleiding willen leveren aan bedrijven, die de tool zouden willen gebruiken. Ook merkte ik dat er weinig updates beschikbaar waren voor deze tool. De reden daarvoor is mij onbekend.

En daardoor kwam ik uiteindelijk tot de constatatie dat volgens mij deze applicatie geen meerwaarde betekent voor mijn stagebedrijf Inuits.

In het kort vat ik even samen waarom ik tot dit besluit durf te komen:

In de eerste plaats zijn enkele functionaliteiten al aanwezig in de huidige infrastructuur van het bedrijf.

Ten tweede wil ik graag wijzen op de complexiteit van de werkwijze : de tool heeft een zeer specifieke omgeving nodig om te kunnen werken. Zodra je 1 kleine aanpassing aanbrengt, wordt het instabiel.

Daarnaast is er zeer weinig documentatie voorhande, voornamelijk over het clusteren van deze omgeving. Hurence, het bedrijf dat deze tool ontwikkelt, brengt ook maar mondjesmaat updates uit. Helaas kon ik de schaalbaarheid niet testen in de container-omgeving, waarin de tool geplaatst stond. Maar met de nodige documentatie en een andere omgeving, lijkt het mij wel haalbaar.

En al ben ik ervan overtuigd dat een medewerker van Inuits tot die zelfde conclusie zou komen na enkele dagen werk, toch ben ik blij met de gekregen tijd. Want de werkervaring, die ik opdeed bij mijn stagebedrijf én het analyseren van zulk een complexe tool, maakte deze stage voor mij ontzettend leerrijk en geeft me de nodige competenties om voluit te gaan voor een carrière in de IT.

5 BRONNEN

- https://github.com/Hurence/logisland
- https://logisland.readthedocs.io
- https://kafka.apache.org/

Faseringstabel

Week 1 & 2	Bekend worden met de werkwijze binnen Inuits.
Week 3 & 4	De applicatie deployen in een lokale virtuele omgeving (Virtualbox).
Week 5 & 6	De Logisland applicatie deployen in de Inuits infrastructuur.
Week 7 & 8	Logs vanuit de echte infrastructuur verwerken in Logisland.
Week 9 & 10	De records die uit Logisland komen analyseren en omvormen tot rapporten.
Week 11 & 12	Demo opzetten waarmee een probleem kan worden opgespoord.
Week 13	Proof of concept en conclusie presenteren aan stagementor.

Plan van aanpak