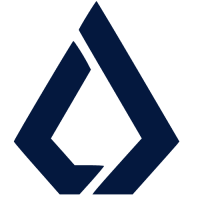
Case study: Observability and Performance

Seven step approach



Project: Lisk Delegates

Project team: BBB-BC

Team: Joeri Berman (2834499)

Luc Urlings (3071081)

Stijn Baltessen (3064565)

Wouter Vandenboorn (3099571)

Floris Feddema (3188256)

Davy de Haas (3099695)

Opdrachtgever: Study case, Bartosz

Versie: 0.1

Versie datum: 21 april 2020

Status: Concept

# Document revisies

|  |  |  |  |
| --- | --- | --- | --- |
| **Versie** | **Wijzigen** | **Auteur** | **Datum** |
| 0.1 | Opzet Document | Stijn Baltessen | 21-04-2020 |

# Inhoudsopgave

[**Document revisies**](#_bu2l4jqkimrr) **2**

[**Inhoudsopgave**](#_nyq276qpg8em) **3**

[**Introductie**](#_eko9gepalqi) **4**

[**Seven step approach**](#_djfvx4apkl6h) **5**

[**1. Bespreek de casus**](#_o18hst4mn4x8) **6**

[**2. Identificeer de vragen**](#_tjp21jqbwrox) **7**

[**3 + 4. Brainstorm & Analyseer**](#_drphanabweja) **8**

[**5. Formuleer de leerdoelen**](#_gbd35qg6f1fh) **9**

[**6. Individuele studie**](#_p83oa97dizn9) **10**

[**7. Discussieer de bevindingen**](https://docs.google.com/document/d/1AkOtzDOreBCDBg6NYK39Q_WG20BLTAAix1-gUmEbzoM/edit#heading=h.a89t2f2kaqr0)[**11**](https://docs.google.com/document/d/1AkOtzDOreBCDBg6NYK39Q_WG20BLTAAix1-gUmEbzoM/edit#heading=h.a89t2f2kaqr0)

# Introductie

# Seven step approach

Hier staan de zeven stappen van de [seven step approach](https://portal.fhict.nl/Studentenplein/LMC/1920vj/Software%20engineering/Afstudeerfase/EnterpriseSoftware/CaseStudies/SevenStepsApproach.docx) methode die bij probleem gebaseerd leren voorkomt.

1. Bespreek de casus
2. Identificeer de vragen
3. Brainstorm over potentiële bekende oplossingen
4. Analyseer resultaat van brainstorm sessie
5. Formuleer de leerdoelen en wat er nog geleerd moet worden
6. Individuele studie om de kennis van 5 te kunnen vergaren
7. Discussieer de bevindingen van 6 en hoe dit 5 en 2 beantwoord

# 1. Bespreek de casus

Hoe zorgt Netflix ervoor dat de applicatie altijd online is?

Wat voor tooling gebruikt Netflix om problemen binnen de applicatie te observeren?

Hoe kan je deze 2 binnen je eigen project toepassen.

# 

# 2. Identificeer de vragen

* Welke observability tooling gebruikt Netflix?
* Wat voor methodes gebruikt Netflix voor hoge uptime?
* Hoe doet Netflix hun applicatie proactief testen zodat ze de applicatie robuust kunnen houden?
* Wat zijn goede observability dashboards en waarom?
* Welke metrics in observability dashboards zijn belangrijk om in de gaten te houden?
* Hoe is het systeem van Netflix distributed?

# 3 + 4. Brainstorm & Analyseer

* Welke observability tooling gebruikt Netflix?

Geen voorkennis.

* Wat voor methodes gebruikt Netflix voor hoge uptime?

AWS, hebben eigen tooling om dit te garanderen. Verder onderzoek.

* Hoe doet Netflix hun applicatie proactief testen zodat ze de applicatie robuust kunnen houden?

Geen voorkennis

* Wat zijn goede observability dashboards en waarom?

Kiali, Grafana

* Welke metrics in observability dashboards zijn belangrijk om in de gaten te houden?

Throughput, uptime, Error codes, response time, huidige load op services.

* Hoe is het systeem van netflix distributed?

Via AWS route 53 CDN

# 5. Formuleer de leerdoelen

De student heeft een goed beeld van wat Netflix doet om hun applicatie te verbeteren.

De student weet wat een observability dashboard is.

De student weet waarom een observability dashboard van belang is.

De student weet wat hoe hij voor het groepsproject observability zou moeten toepassen.

# 6. Individuele studie

* Welke observability tooling gebruikt Netflix? - Luc & Stijn
* Wat zijn goede observability dashboards en waarom? Luc & Stijn
* Wat voor methodes gebruikt Netflix voor hoge uptime? Wouter & Floris
* Hoe doet Netflix hun applicatie proactief testen zodat ze de applicatie robuust kunnen houden? Wouter & Floris
* Welke metrics in observability dashboards zijn belangrijk om in de gaten te houden? Davy & Joeri
* Hoe is het systeem van Netflix distributed? Joeri & Davy

## Welke observability tooling gebruikt Netflix?

Netflix gebruikt een variatie van tooling die allemaal hun eigen doeleinde hebben, we hebben deze kunnen halen uit <https://netflix.github.io/>.

Voor een at glance observering gebruikt netflix: [vizceral](https://github.com/Netflix/vizceral). zodat je snel een model voor je hebt met het observeren van je infrastructuur.

Netflix gebruikt [edda](https://github.com/Netflix/edda) om veranderingen binnen hun cloud omgeving bij te houden.

Netflix gebruikt [spectator](https://github.com/Netflix/spectator) om dimensional [Time series](https://en.wikipedia.org/wiki/Time_series) op te nemen.

Netflix gebruikt [Atlas](https://github.com/Netflix/atlas) om time series data te verwerken en bewerken.

***spectator neemt de data op, atlas verwerkt de data.***

Netflix maakt gebruikt van [Vector](https://github.com/Netflix/vector) voor de weergave van de data zodat ze hier insight op krijgen.

## 

## 

## Wat zijn goede observability dashboards en waarom?

Netflix gebruikt Vector deze hebben zij zelf ontwikkelt, maar er zijn natuurlijk nog vele andere alternatieven. We willen hier kort op ingaan om licht te geven aan de alternatieven. Zodra er grote voordelen direct zichtbaar zijn zullen we deze beschrijven. Deze vraag gaat dus vooral in op de weergave van de dashboards en niet de verzameling van data.

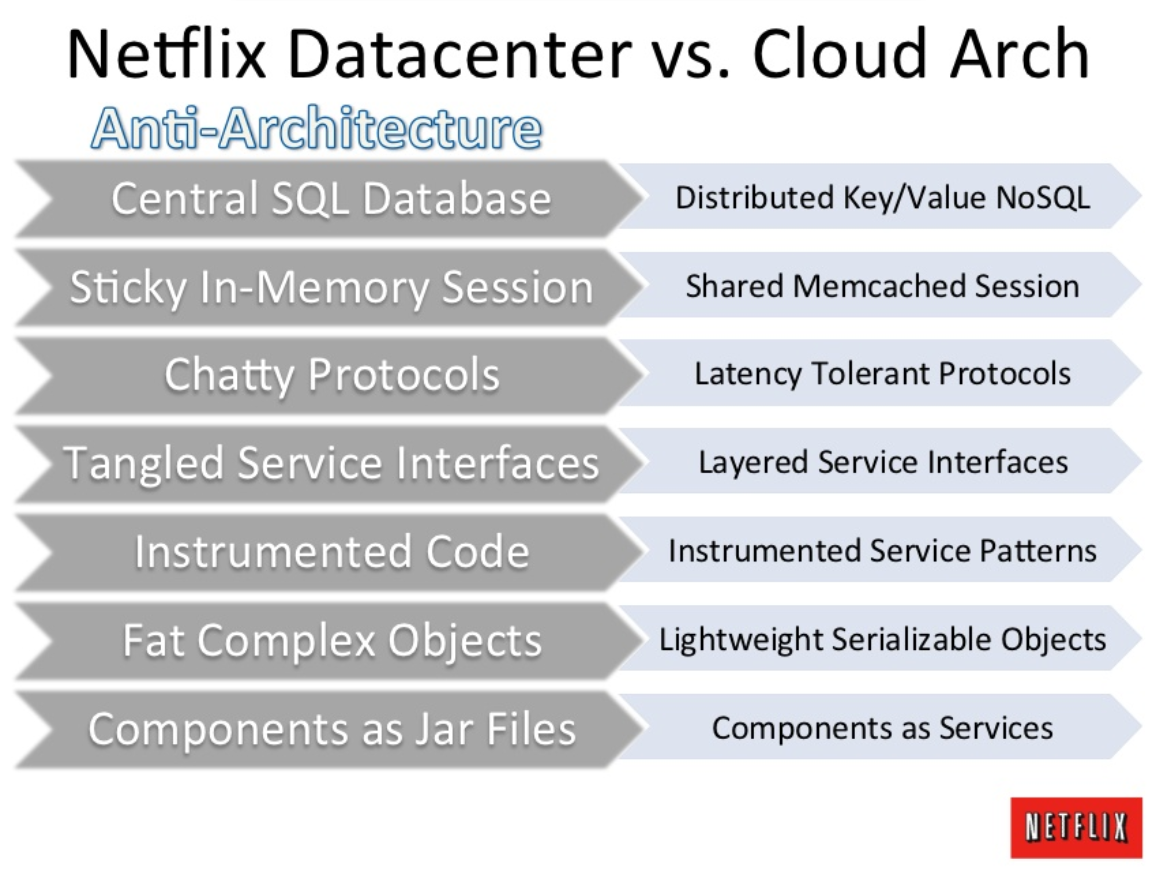
|  |
| --- |
|  |

* Kibana. Dit is een open source browser based analytics dashboard voor elasticsearch
  + zeer mature
  + Querying and searching logs is een van de sterkte punten van kibana.
  + Hoog gebruik 17.000+ commits.
  + Hoog interest
* Grafana. Dit is een general purpose observability dashboard and graph composer.
  + Fun fact: “Grafana began as a fork of Kibana.”
  + hoog gebruik 14.000+ commits
* Kiali. Dit is een observability console voor istio met service mesh configuratie mogelijkheden.
* Graphite. Deze tool doet eigenlijk twee dingen. Het slaat de data op en maakt er grafieken van.
* Cacti, maakt het mogelijk om netwerk grafieken te maken.

## Wat voor methodes gebruikt Netflix voor hoge uptime?

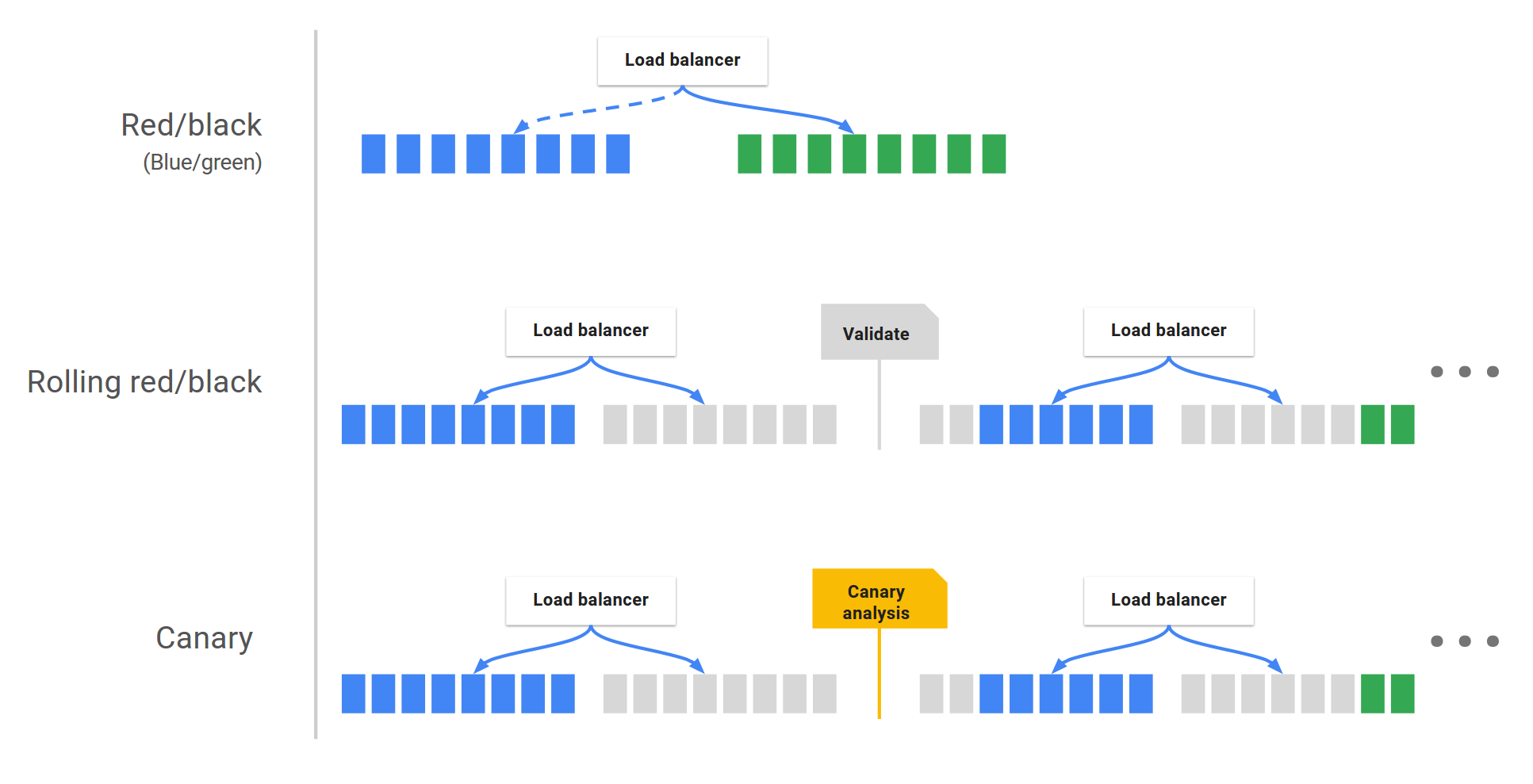
Netflix gebruikt AWS met 12 verschillende regio's die elk weer bestaat uit een aantal verschillende availability zones. Deze zones heeft minimaal een datacenter met stroom en netwerk. Doordat deze zones met elkaar verbonden zijn kan er communicatie plaatsvinden tussen applicaties in de verschillende zones zodat uitval van een van deze zones opgevangen kan worden.

Netflix heeft ervoor gekozen om over de jaren heen langzaam naar een Cloud Native architectuur over te gaan die draait in de AWS Cloud. Door hierbij ook goed gebruik te maken van Cloud Architecture Patterns en veel te investeren om de services van netflix op veel verschillende fysieke locaties te draaien is het mogelijk om een hogere uptime te krijgen.

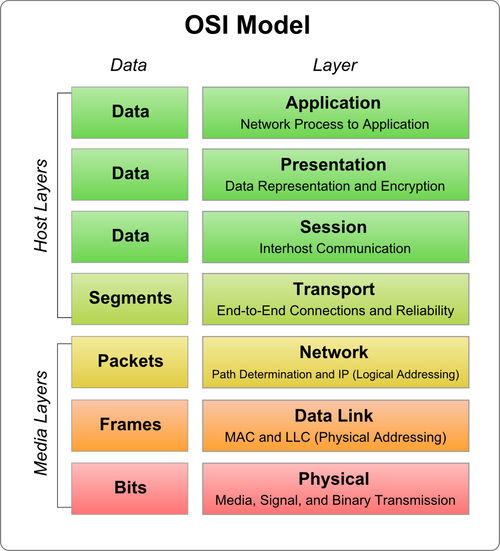


Spinnaker is de CD tool die Netflix heeft gemaakt om te deployen naar AWS, het kan multi-cloud draaien en het is eenvoudig om applicaties op verschillende soorten te deployen.

Zo zijn er een aantal verschillende deployment strategieën die Netflix gebruik met Spinnaker zoals hieronder weergegeven

Een lijst van een aantal tools van Netflix die gebruikt worden:

* Eureka; een REST based mid-tier load balancer die gebruikt wordt in de AWS cloud
* Archaius; een Java library voor configuration management
* Ribbon; een Java library voor remote procedure calls voor load balancers
* Hystrix; een latency en fault tolerance Java library voor gedistribueerde systemen
* Zuul; Een L7 applicatie gateway met dynamic routing



## Hoe test Netflix proactief zodat ze de applicatie robuust kunnen houden?

Netflix maakt hiervoor gebruik van een set van zelf ontwikkelde tools waarmee ze beschikbaarheid, betrouwbaarheid, beveiliging en veerkrachtigheid van hun AWS infrastructuur kunnen toetsen. Ze noemen deze set van tools de **Simian Army**.

Deze tools zijn beschikbaar via : [**https://netflix.github.io/chaosmonkey/**](https://netflix.github.io/chaosmonkey/)

Dit in combinatie met het gebruik van Devops zorgt ervoor dat Netflix een zeer robuuste omgeving heeft waarin hun services worden geleverd.

**Chaos monkey**

"Imagine a monkey entering a 'data center', these 'farms' of servers that host all the critical functions of our online activities. The monkey randomly rips cables, destroys devices and returns everything that passes by the hand [i.e. flings excrement]. The challenge for IT managers is to design the information system they are responsible for so that it can work despite these monkeys, which no one ever knows when they arrive and what they will destroy."

* Chaos Kong  
  Dropt een volledige AWS Region. Het is zeldzaam dat dit gebeurt maar op deze manier kan er wel gewerkt worden aan mogelijkheden om de schade te beperken en reactie en herstel te toetsen in een soortgelijke situatie die niet gesimuleerd is.
* Chaos Gorilla

Dropt een volledige amazon “availability zone” ofwel een of meerdere datacenters in een geografische regio.

* Latency Monkey

Simuleert communicatie vertragingen binnen het netwerk.

* Doctor Monkey

Monitort “health checks”, hiermee worden metrics zoals CPU load en andere performance metrics bijgehouden.

* Janitor Monkey

Identificeert ongebruikte resources om zo “waste and clutter” te identificeren en op te ruimen.

* Conformity Monkey

Een tool die bepaalt of een instantie niet voldoet door deze te testen aan de hand van een reeks regels. Als een van de regels bepaalt dat de instantie niet conform is, stuurt de monkey een e-mail naar de eigenaar van de instantie.

* Security Monkey

Afgeleid van Conformity Monkey, een tool die instanties zoekt en uitschakelt met bekende kwetsbaarheden of onjuiste configuraties.

* 10-18 Monkey

Een tool die problemen met lokalisatie en internationalisatie detecteert.

* Byte-Monkey

Een kleine Java-library voor het testen van faalscenario's in JVM-toepassingen. Het werkt door code on-the-fly te instrumenteren om opzettelijk fouten zoals uitzonderingen en latentie te introduceren.

* Chaos Machine

ChaosMachine is een tool die chaos engineering doet op applicatieniveau in de JVM. Het concentreert zich op het analyseren van de error-handling capability van elk try-catch-blok dat bij de toepassing betrokken is door uitzonderingen te injecteren.

* Gremlin

Een 'failure-as-a-service'-platform dat is gebouwd om het internet betrouwbaarder te maken. Door testomgevingen beschikbaar te maken kunnen dit soort services getest worden zonder dit op een live omgeving te hoeven doen.

* Facebook Storm

Vergelijkbaar met de Chaos Gorilla, dit is de facebook implementatie hiervan.

* Days of Chaos

Gamification vanuit AWS GameDays. Elke 30 minuten simuleerden operators storingen in de pre-productie. Teams verdienen punten op basis van detecties, diagnoses en oplossingen.

* ChaoSlingr

Voert Cyberaanvallen op AWS infrastructuur uit om zo zwakke punten te ontdekken binnen de complexe gedistribueerde systeem omgevingen.

* Chaos Toolkit

Wordt gebruikt om aan te tonen dat de experimentatie approach op verschillende niveaus kan worden gedaan: infrastructuur, platform maar ook applicatie.

* Mangle

Een geautomatiseerde tool om met zo min mogelijk voor configuratie testing mogelijk te maken op bekende tools en services. denk hierbij aan K8S, Docker, vCenter en virtual machines.

* Chaos Mesh

Een tool die load balancing op een kubernetes omgeving kan testen.

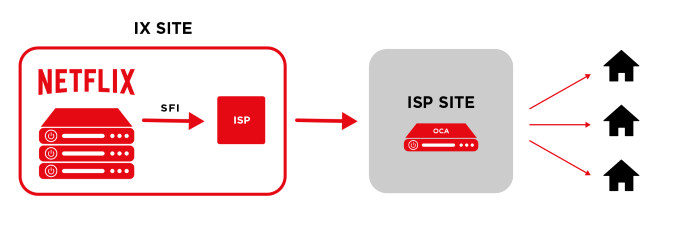
* Litmus Chaos

Litmus is een toolset voor cloud-native chaos engineering. Litmus biedt tools om chaos op Kubernetes los te laten om zo SRE's te helpen zwakke punten in hun implementaties te vinden.

## Welke metrics in observability dashboards zijn belangrijk om in de gaten te houden?

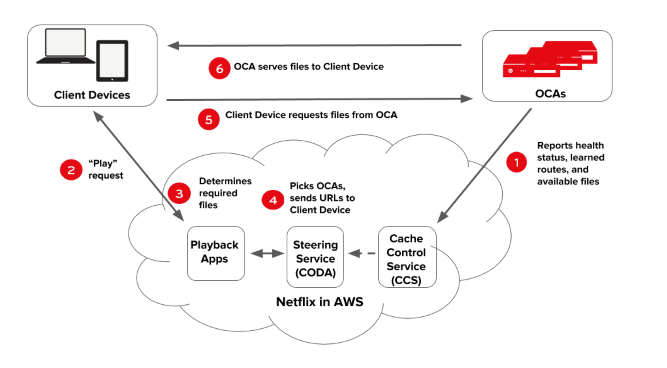
* Bandbreedte - De maximale hoeveelheid data die per seconde kan worden verstuurd binnen het netwerk. Zodra dit zijn limiet bereikt zal er een uitbreiding nodig zijn zodat het ophalen van data zo snel mogelijk kan gebeuren en er geen extra tijd benodigd is wat ten koste gaat van het gebruiksgemak van een service.
* Throughput - De hoeveelheid data per seconde die daadwerkelijk wordt verstuurd. Dit wordt ook wel genoteerd als de percentage van datapakketten die succesvol worden verstuurd. Een lage throughput betekent dat er veel pakketten opnieuw moeten worden verzonden.
* Retransmission - Als data pakketten niet aankomen, moet het netwerk ze opnieuw versturen. De tijd die hiervoor nodig is laat zien hoe lang het duurt voor het netwerk om hiervan te herstellen. De retransmission ratio laat weten hoe vaak pakketten niet aankomen en laat zien of er een opstopping is binnen het netwerk.
* Latency - De vertraging tussen een input en response in de dataoverdracht. Vaak is latency niet te zien met het menselijke oog, daarom is het goed om dit te monitoren voor vertragingen die ontstaan. Bij audio toepassingen noem je dit jitter.
* Availability - Dit is belangrijk om te weten of het netwerk beschikbaar is. Je kunt nooit 100% uptime garanderen, maar je wilt wel weten wanneer het netwerk offline is en hoeveel procent uptime er wordt gehaald. Als het netwerk offline is zijn monitors belangrijk en kunnen je meteen waarschuwen zodat je zo snel mogelijk kan handelen.
* Connectivity - Er moet duidelijk zijn of er wel of geen connecties zijn tussen de nodes binnen een netwerk.
* Soort data - Het kan belangrijk zijn om te weten welke soort data er wordt verwacht van een cliënt. Als bijvoorbeeld een bepaalde film veel wordt gekeken moet het netwerk meer resources geven aan de service die deze film beschikbaar maakt zodat er geen problemen ontstaan.

## Hoe is het systeem van Netflix distributed?

Netflix maakt gebruik van een systeem dat zij Open Connect noemen. Dit systeem zorgt voor de datastroom van Netflix naar de ISP’s. Het systeem bestaat uit Open Connect Appliances die ervoor zorgen dat de data zo snel mogelijk naar de gebruiker gaan. Deze apparaten worden door netflix geleverd aan de ISP’s die deze vervolgens in hun netwerk kunnen opnemen.

De data die de OCA doorsluizen naar de gebruikers komen van een AWS cloud oplossing. Hier worden alle data opgeslagen en Amazon zorgt ervoor dat de data verspreid over de wereld wordt opgeslagen met hun AWS route 53 CDN.

De OCA’s werken ook als een soort van buffer voor de AWS omgeving. Hierdoor kan veel opgevraagde data snel worden verwerkt en doorgestuurd naar de gebruiker.



# 7. Discussieer bevindingen

De gediscussieerd leerdoelen zijn behaald.

# Conclusie

Met de informatie die we hebben opgedaan hebben we een goed inzicht gekregen van het proces binnen Netflix over het onderhouden van het systeem door middel van logging en monitoring.

Voor ons eigen project zouden we het volgende kunnen indien we hier tijd voor hadden:

* Een dataverzamelaar implementeren van alle logging.
* Een dataverwerker implementeren van alle data.
* Een weergave dashboard implementeren (Grafana) op basis van de genoteerde metrics.

Om de performance te waarborgen zouden we het volgende kunnen doen voor de groepsopdracht:

* chaos monkeys implementeren, echter het probleem hiermee is dat we een te klein netwerk hebben om dit te testen.