Advies over de communicatie tussen de groeps-servers

De centrale bank

Paul de Hek
0941736
Ti1C

Inhoudsopgave

Inleiding

In dit rapport zal ik bespreken hoe je op een efficiënte manier kan communiceren tussen meerdere groepsservers. Deze oplossing moet voldoen aan de kwaliteitseisen die ik heb opgesteld, deze zijn terug te vinden een document op mijn website (www.stud.hr.nl/0941736). Hier zijn ook mijn risicolog en issuetracker te vinden.
Ook heb ik nog 2 niet functionele eisen gesteld aan dit verslag, dit zijn:

- Het document heeft een gestructureerde en logische volgorde
- Het document heeft een fijne lay-out en is prettig om te lezen.

Hoewel de originele vraag was “Hoe richt ik efficiënte manier de centrale bank in?” heb ik dit uitgebreid naar “Hoe communiceer ik op een efficiënte manier met andere servers?” omdat er andere mogelijkheden zijn waar ik ook naar wilde kijken.

Analyse

Het protocol

Om ze zorgen dat banken met elkaar kunnen communiceren zal er eerst gezorgd moeten worden dat alle groepsservers op dezelfde manier met elkaar proberen te communiceren en de zelfde soort data versturen en ontvangen ik zal hier bespreken wat goede opties zijn

Message Queuing

Er moeten berichten doorgeven worden van de ene server naar de andere server hier is Message Queuing heel geschikt voor. Met Message Queuing geeft een zogeheten broker de berichten door van de ene client naar de andere, deze kan dan op zijn eigen tempo deze berichten verwerken. Een situatie waarin dit heel handig is is als er een heleboel verzoeken komen met Message Queuing raakt de server niet overbelast maar zal hij de berichten normaal een voor een afwerken. Het kan zijn dat je als klant langer dan normaal moet wachten tot te ingelogd bent maar met een ander protocol zou het kunnen zijn dat je helemaal niet ingelogd wordt omdat de connectie afgebroken word vanwege een overbelastte server.

Ook is het schaalbaar als omdat als er steeds meer dataverkeer komt kan je een extra broker toevoegen om de data beter naar de goede ontvanger te sturen.

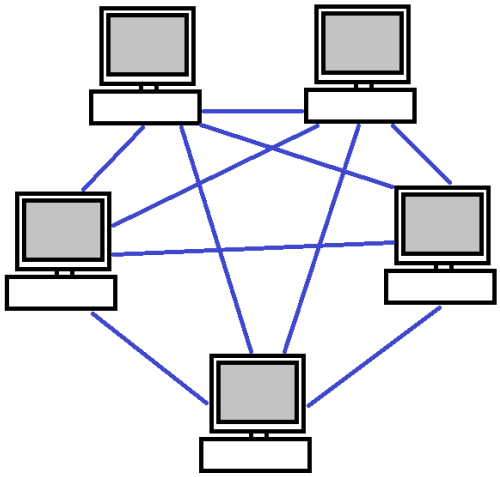
Er zijn verschillende soorten Message Queuing de twee die ik zal behandelen zijn Message Queuing Telemetry Transport (MQTT) en Advanced Message Queuing Protocol (AMQP). Het idee achter de werking is bij beide protocollen hetzelfde, het zijn immers beide variaties op het Message Queuing protocol. Het grootste verschil zit hem in hoe de data verstuurd wordt, AMQP gebruikt namelijk een HTTPS. Hierdoor heeft AMQP een beveiligde verbinding maar dit zorgt ook voor een grote header en meer data die verstuurd word dan met MQTT. MQTT gaat namelijk direct over de TCP laag en is hierdoor niet beveiligd maar heeft hierdoor heeft het ook een veel kleinere header waardoor er minder data te versturen is.

Beide protocollen hebben een Quality-of-Service, dit houd in dat je kan instellen hoe groot de controle is met het sturen. QoS 0 houdt in dat je gewoon een bericht stuurt en dan moet je eigenlijk maar hopen dat het aankomt. Bij QoS 1 zal de ontvanger een bericht terug sturen als hij het bericht heeft ontvangen. Qos 2 is het meest veilig als je zeker wil weten dat je bericht aankomt aangezien je hier een gehele four-way handshake hebt

Het netwerk

Peer-to-peer netwerk:

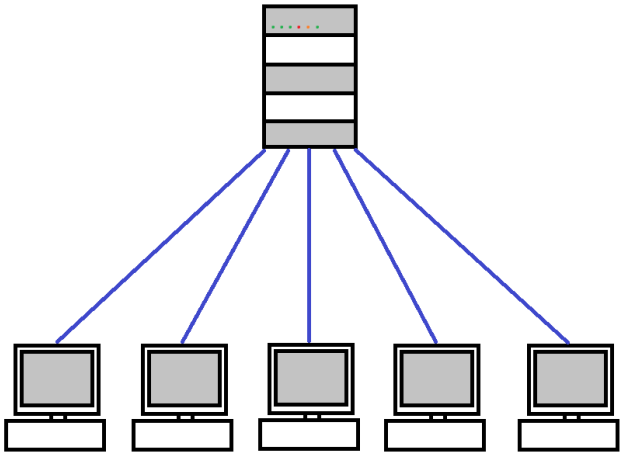
In deze vorm van communicatie zijn de servers van elke bank verbonden met alle andere servers van de andere banken. Hieronder is dit schematisch weergegeven.



In deze oplossing zijn alle groepsservers (hierna servers) direct verbonden. Dit moet uiteraard veilig gebeuren en efficiënt gebeuren. de grootste uitdaging zal zijn verschillende groepen verschillende programmeertalen hebben gebruikt. Er zijn twee manieren waarop dit opgelost kan worden. De eerste is dat er één programmeertaal word gekozen waarin de servers met elkaar communiceren. Dit is waarschijnlijk de taal waarin de meeste servers geschreven zijn (in dit project zal dit Java zijn), hierdoor zal een deel van de servers niks hoeven te vertalen maar een server die in een andere taal is geschreven, zoals C, zal zowel bij het verzenden als het ontvangen de data moeten omzetten in zijn taal waardoor deze bank langzamer zal werken. De tweede oplossing is dat elke server in meerdere talen kan data kan ontvangen en dit zelf omzet naar de taal die binnen de server word gebruikt. Er zal een afweging gemaakt moeten worden aangezien bij de eerste oplossing er alleen werk zal moeten worden verricht als de server niet is geschreven in de afgesproken communicatie taal en is er voor een deel van de servers geen extra vertaal werk nodig. Bij de tweede oplossing is voor iedereen verzenden even snel omdat iedereen data verzend in zijn eigen taal maar het ontvangen zal langer duren. Ook is dit voor iedereen meer werk omdat elke server elke programmeertaal moet kunnen ontvangen die in het systeem word gebruikt.
Security is bij een peer-to-peer netwerk lastig omdat er zoveel verbindingen zijn die allemaal beveiligd moeten worden. Als er TLS gebruikt word zal dit over elke connectie gebruikt moeten worden. Dit is mogelijk maar het is niet heel erg praktisch om te doen.

Centrale server:

Bij deze vorm van communicatie zijn alle servers verbonden met één centrale server die de data tussen de servers verstuurd. Dit is hiernaast weer schematisch weergegeven.



Bij dit model zullen de servers verbinding maken met een centrale server die de data heen en weer stuurt tussen de goede servers. Een scenario kan als volgt gaan:

Een server leest een bankpas die hij niet in zijn database heeft staan, hij zal dan naar de centrale server een bericht verzenden om te kijken of de bankpas in een database van een andere server staat. De centrale server zal dan een bericht sturen naar alle andere servers. Als een server terug stuurt dat hij de pas herkent zal de centrale server dit terug sturen naar de originele server. Ook zal de centrale server doorsturen bij welke bank de pas hoort.

Het scenario wat ik hierboven heb beschreven is de eerste interactie tussen een bepaalde bankpas en de server.
Bij deze vorm van communicatie is hetzelfde probleem aanwezig als bij het peer-to-peer netwerk. De servers zijn in verschillende talen geschreven. Het is nu alleen makkelijker om alle servers in hun eigen taal de data te laten sturen aangezien er maar een ontvanger is, de centrale server. Deze word waarschijnlijk in Java geschreven en er zullen klasses zijn om de data uit verschillende talen om te zetten naar iets wat de centrale server kan lezen. Ook zullen deze klasses de data weer terug omzetten zodat de server de data in zijn eigen taal ontvangt. Deze methode is ook via TLS te beveiligen wat ook de bedoeling is om doen

Vergelijken

Er zijn uiteraard verschillen tussen deze twee manieren van verbinden ik zal hier de meest belangrijke verschillen behandelen en de voor- en nadelen ten opzicht van elkaar

Een peer-to-peer netwerk heeft als voordeel dat de communicatie in theorie iets sneller is omdat de data direct van de ene server naar de andere server gaat. Bij een centrale server gaat dit nog door een extra server die dan nog een keer extra de data moet verwerken waardoor het langzamer is.
Dit heeft wel een keerzijde waardoor een peer-to-peer netwerk in praktijk zelfs langzamer kan zijn dan een centrale server gebruiken. Dit komt omdat bij een peer-to-peer netwerk de server die de request doet zelf alle informatie moet verwerken om te kijken met welke server hij verder moet communiceren. Het is makkelijker om dit op een centrale server te doen die alleen maar de data moet doorgeven van de ene server naar de andere. Een centrale server kan een deel van de last op zich nemen waardoor de andere servers bezig kunnen houden met de bankautomaten

Bij een peer-to-peer netwerk heb je ook meer controle over met wie je welke data deelt dan met een centrale server. Bij een centrale server krijg je namelijk alleen een aanvraag om bepaalde data te verzenden of te verifiëren, dat stuur je terug naar de centrale server en die bepaalt waar de data naar toe gaat. Bij een centrale server heb je dus geen controle over hoe en naar wie jouw data verstuurd wordt. Bij een peer-to-peer netwerk kan je zelf bepalen naar wie en welke dat je verstuurd wat je dus meer controle geeft.

Een centrale server geeft echter meer schaalbaarheid voor het gehele netwerk. Dit komt doordat het redelijk eenvoudig is om een extra server met de centrale server te verbinden omdat er maar een extra verbinding aangelegd moet worden. Bij een peer-to-peer netwerk moet er een verbinding met elke andere server in het netwerk gemaakt worden. Dit wel veel meer moeite.

Advies

Het protocol

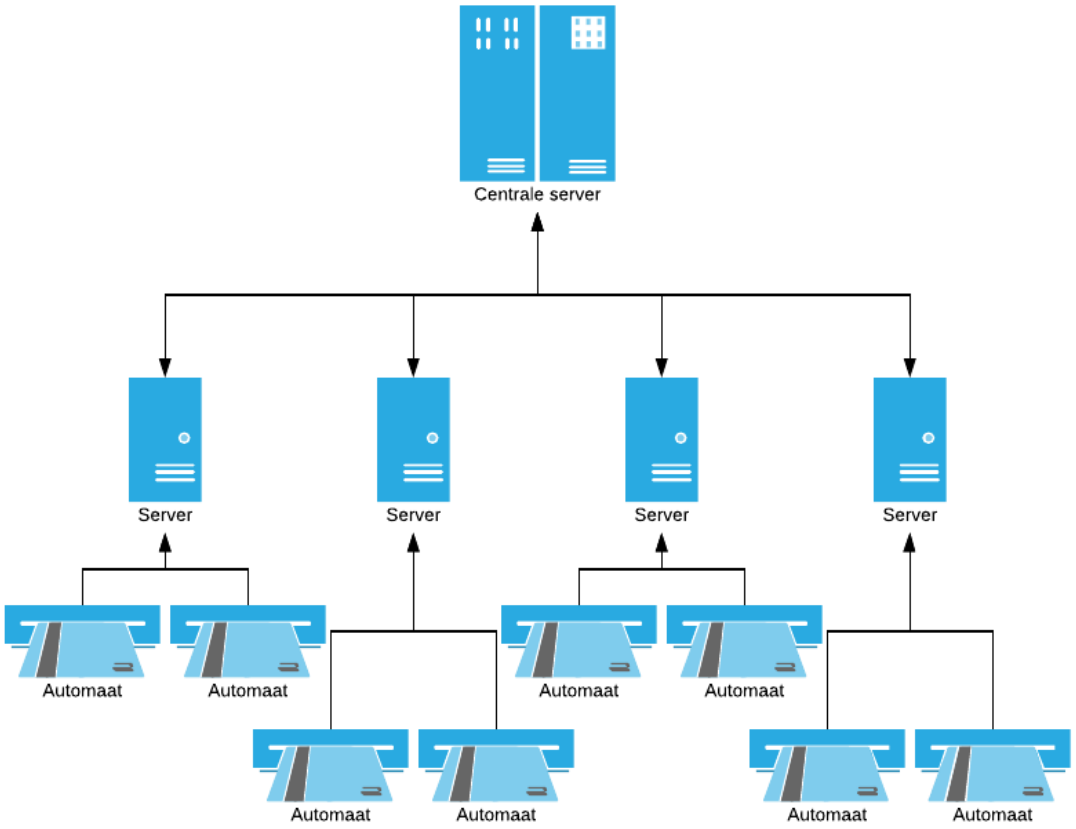
Om de data te versturen wil MQTT gebruiken over een TLS verbinding. Mijn reden hiervoor is vooral de kleinere packet size van MQTT tegenover AMQP, het is namelijk heel belangrijk dat de klant snel alles kan regelen en zo min mogelijk tijd wacht. De TLS die gebruikt is om de berichten veilig te houden. Het was voor mij best belangrijk om TLS te gebruiken omdat toen we gingen rondvragen het bleek dat veel groepjes TLS al geïmplementeerd hebben voor de communicatie tussen hun server en bankautomaten of dit willen doen. Zij hebben hier al kennis over en het zal het makkelijker maken om in de korte tijd die we hebben een werkende verbinding op te zetten met het netwerk.
Om alles veilig te houden zal ook alle inlog data voor verzenden naar de andere servers gehasht worden en gehasht opgeslagen worden in de databases. Dit is ervoor om te zorgen dat niemand achter onder andere de pincode van een pas kan komen.
De enige data die verzonden zal worden is de data die de gebruiker zelf invoert of die op het gescherm getoond moet worden. Alle authenticatie en checks worden door de server van de bank van de gebruiker gedaan. Dit is om alle informatie zo veilig mogelijk te houden. De ene server stuurt bijvoorbeeld dus een saldo en een rekeningnummer met de aanvraag om geld op te nemen. De andere server checkt of er genoeg geld op de rekening staat schrijft dit af als de gebruiker genoeg saldo heeft en zal een true terug sturen naar de automaat via de servers zodat de automaat het geld kan uitgeven.
Om het hele netwerk zo snel mogelijk te laten werken zal elke server ook een eigen adressentabel bijhouden in de database. In deze database staan een gehashte UID en een bytegroep aan elkaar gekoppeld. Als de server een UID binnenkrijgt zal hij eerst in zijn eigendatabase kijken of het een klant van deze bank is. Zodra dit niet het geval is zal hij in de adressentabel kijken of hij daar de UID kan vinden. Als de UID in de adressentabel staat zal de server een gericht bericht naar de desbetreffende server sturen voor de data die hij wil hebben of wil laten checken. Dit is handig omdat dan niet alle servers gevraagd worden om te kijken of zij die data hebben. Er zal alleen een bericht naar alle servers worden gestuurd als de server de UID nog niet kent en hem dus niet in zijn adressentabel heeft staan. Hierna zal hij deze UID toevoegen aan zijn adressentabel en later dus gericht een bericht kunnen sturen naar de server van de groep waar deze pas bij hoort. Hierdoor blijft het netwerk snel maar nog wel veilig.

Het netwerk

De communicatie tussen de servers zal plaatsvinden met behulp van een centrale server. Ik heb hiervoor gekozen omdat dit de handigste manier is om MQTT toe te passen. De centrale server dient hier als broker. Ook is het belangrijk dat het netwerk schaalbaar is wat zoals eerder genoemd beter te doen is met een centrale server.
Er is minder controle over jouw data maar dit zou geen heel groot probleem moeten zijn aangezien er duidelijke afspraken zijn over hoe alles verstuurd wordt en afgehandeld wordt. Er is dus weinig controle nodig. Ook is het prettig om zoveel mogelijk last van de servers af te houden en die last te verplaatsen naar de centrale server. Daarom zal elke server in zijn eigen taal werken en verzenden en doet de server het vertaalwerk, zo kunnen de servers zich bezig houden met het verwerken van de data voor de beste ervering voor de gebruiker.

Op de volgende pagina staat een netwerk diagram en een dataflow diagram. Deze zijn ook te vinden op mijn website.

Netwerkdiagram



Dataflow diagram