Gedistribueerde Systemen

Memory Assignment

Door:

Thomas Bruneel (MELICTISW)
Birgen Vermang (MELICTISW)

Academiejaar 2018-2019
Master Industrieel Ingenieur
KU Leuven Technologiecampus Gent

Inhoudsopgave

1. Architectuur

Onze applicatie bevat volgende instanties: dispatcher, databankserver, applicatieserver en client. Initieel wordt de dispatcher gestart. Deze start vervolgens de 3 databankservers en 1 applicatieserver. Bij de initialisatie koppelt de dispatcher de databankservers onderling en wordt de applicatieserver met een databankserver gekoppeld.

Wanneer we later meer applicatieservers nodig hebben passen we loadbalancing toe. De dispatcher zal een applicatieserver koppelen met de databankserver die het minst gelegde connecties heeft. Hierdoor wordt elke server optimaal gebruikt en zorgen we ervoor dat de processing eerlijk verdeeld blijft.

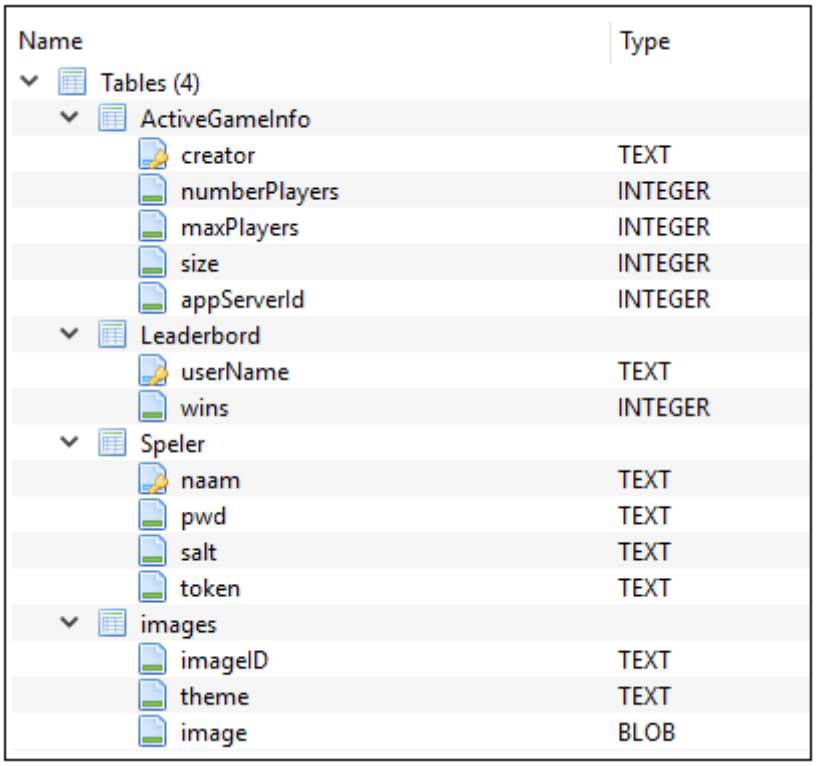
Wanneer een client zich aanmeldt bij de dispatcher, zal de dispatcher die client koppelen aan een applicatieserver, rekening houdend met het aantal actieve games per applicatieserver. Meer precies wordt de client gekoppeld met de applicatieserver die op dat moment het minst aantal games host. Wanneer elke applicatieserver die reeds bestaat het maximum toegelaten aantal games host, wordt er een nieuwe applicatieserver gestart en de client wordt hieraan toegewezen.

Games worden lokaal op een applicatieserver gehost. In de lobby worden alle games weergegeven die op elke applicatieserver gehost worden.
Wanneer de client ervoor kiest om zelf een spel te starten dan host de huidige server dit spel. Wanneer wordt gekozen om een spel te joinen of te spectaten, wordt de client verplaatst naar de applicatieserver waar het spel gehost wordt.

Om het ontwikkelingsproces te versnellen hebben we alle functionaliteiten van applicatieserver, databankserver en dispatcher elk in één service gestoken. Bij een volgende iteratie zouden deze services moeten worden opgedeeld in aparte services. Bijvoorbeeld: Een aparte appserverservice voor de dispatcher en de client.

2. Databank

Als databasemanagementsysteem hebben we gebruik gemaakt van SQLite.
Onze databank bevat volgende tabellen met bijhorende attributen:



We kunnen de naam van de speler gebruiken als primary key omdat we eisen dat de gebruikersnaam uniek is.

De *appServerId* bij *ActiveGameInfo* zorgt ervoor dat we weten welke game op welke applicatieserver staat. De appServerId komt overeen met het poortnummer van de applicatieserver die de game host. Zo kan de client makkelijk met de juiste applicatieserver verbonden worden wanneer hij een game joint of spectate in de lobby.

3. Ontwerpbeslissingen

3.1 Consistentie en replica

De databankservers zijn onderling met elkaar verbonden waardoor ze onderling kunnen communiceren met elkaar. Wanneer een databankserver gegevens update in zijn eigen databank, zal deze databakserver deze update ook persisten naar de andere databankservers. Zo voeren ze deze update ook door in hun eigen tabellen. Hierdoor bevat iedere databank op ieder moment dezelfde data.

3.2 Caching

De afbeeldingen van onze kaarten worden opgeslagen in de databank. Wanneer een client inlogt wordt een verzoek van de appserver naar de databankserver gestuurd. De databankserver haalt de afbeeldingen uit zijn databank en verzendt deze naar de applicatieserver en vervolgens naar de client, waar hij ze nadien lokaal opslaat. Dit voorkomt onnodige overhead omdat we niet elke keer afbeeldingen moeten verzenden als een client tijdens zijn beurt een kaart omdraait.

De staat van elk spel wordt zowel op de server als op elke client bijgehouden. Dit zorgt voor een kleine hoeveelheid overhead, maar de gecachte data kan bij een volgende iteratie gebruikt worden voor recovery.

3.3 Beveiliging

Wanneer een client registreert, moet het wachtwoord aan volgende vereisten voldoen:

- Minstens 8 karakters
- Minstens één speciaal karakter
- Minstens één kleine letter
- Minstens één hoofdletter
- Minstens één getal

Deze vereisten worden aan client side gecontroleerd zodat onnodige overhead wordt vermeden. Indien aan alle vereisten voldaan zijn, wordt de naam doorgestuurd naar de applicatieserver waar gecontroleerd wordt of de username nog niet in gebruik is.
Wanneer hier ook aan voldaan is wordt op de applicatieserver een salt gegenereerd en wordt deze gehashed met het wachtwoord. Nadien wordt het gehashte wachtwoord en salt opgeslagen in de databank. Om onze wachtwoorden te hashen gebruiken we bcrypt.
Wanneer een client inlogt, wordt de salt uit de database gehaald, gehashed met het meegegeven wachtwoord en vergeleken met het gehashte wachtwoord uit de databank.

Als het inloggen succesvol is zal een token worden aangemaakt. Een token heeft een levensduur van 24 uur. Bij elke navigatie naar een andere pagina, wordt gekeken of deze 24 uur al overschreden is. Indien dit het geval is wordt de client uitgelogd.
De applicatieserver heeft een secret key waarmee hij de tokens kan signen met het SHA-256 algoritme. Zo kunnen we er vanuit gaan dat tokens niet worden gewijzigd tijdens transmissie.

3.4 Recovery

Als een databankserver zou uitvallen, leidt dit niet tot verlies van gegevens. Dit komt doordat gegevens en updates telkens gepersist worden tussen alle databanken onderling.

We hebben geen recovery strategie geïmplementeerd in onze code, maar hebben wel beginselen geschreven voor de volgende strategie die het uitvallen van 1 databank toelaat:

Elke databankserver bevat twee queues die gevuld kan worden met updates voor de uitgevallen databank. De eerste queue bevat nieuwe gebruikers die zich registreerden in de periode waarin de databank offline was. De tweede queue houdt de informatie bij van de games die momenteel gehost worden door de applicatieservers waarmee de databankserver verbonden is.

Wanneer de uitgevallen server terug online komt haalt hij eerst de data uit alle queues van de andere databanken. Omdat een databank enkel info in zijn queues steekt die niet tot bij de uitgevallen databankserver zijn geraakt maar wel tot bij de andere databank die nog steeds online is, zal er geen duplicate data voorkomen.

De dispatcher zal de applicatieservers die verbonden waren met die uitgevallen databankserver opnieuw laten connecteren met een andere applicatieserver. Hierbij wordt ook weer loadbalancing toegepast.

Wanneer een applicatieserver uitvalt gaat alle data op de applicatieserver verloren, maar de informatie van het spel dat aan de gang was wordt gecached bij de clients. Bij een volgende iteratie zouden we met deze gecachte data een recovery strategie kunnen uitwerken.

4. Overzicht API's

Zie Javadocs op github.

5. Kritische reflectie

5.1 Sterkte

- Goede loadbalancing van de applicatieservers en databankservers.
- Games worden lokaal op de appserver gespeeld en niet gepersist naar de databank. Hierdoor worden de databankservers niet belast met het opslaan van elke move.

5.2 Zwakte

- Wanneer een client een game verlaat, wordt de game voor de andere spelers in die lobby ook beëindigd.
- Applicatieservers worden nooit afgesloten. Een applicatieserver waar geen actieve games op gehost worden zal door de loadbalancing wel snel gevuld worden met clients die een spel kunnen hosten.

5.3 Uitbreidingen

Indien we meer tijd hadden, zouden we volgende features toevoegen:

- Wanneer een client een game verlaat, zouden de andere clients moeten kunnen verder spelen.
- De besproken recoverystrategieën implementeren.
- Onze grote services splitsen in microservices.

6. Link naar project

https://github.com/thomasbruneel/opdracht_DS/tree/dispatcher