Reflectieverslag

WIN Load Balancer

Naam: Yigit Baylan  
Studentnummer: 601714  
Docent: Martijn Driessen  
Semester: NOTS  
Course: WIN  
Plaats: Arnhem  
Datum: 6 april 2020

Inhoudsopgave

[1. Inleiding 2](#_Toc36923627)

[2. Ontwerpkeuzes 3](#_Toc36923628)

[2.1. Load verdeling 3](#_Toc36923629)

[2.1.2. Codevoorbeeld 3](#_Toc36923630)

[2.1.3. Alternatieven 3](#_Toc36923631)

[2.1.4. Bronnen 3](#_Toc36923632)

[2.2. Persistentie 4](#_Toc36923633)

[2.2.2. Codevoorbeeld 4](#_Toc36923634)

[2.2.3. Alternatieven 4](#_Toc36923635)

[2.2.4. Bronnen 4](#_Toc36923636)

[2.3. Health monitoring 5](#_Toc36923637)

[2.3.2. Codevoorbeeld 5](#_Toc36923638)

[2.3.3. Alternatieven 5](#_Toc36923639)

[2.3.4. Bronnen 5](#_Toc36923640)

[3. Beroepsproduct 6](#_Toc36923641)

[4. Conclusie 7](#_Toc36923642)

# Inleiding

Voor de WIN-course moet een loadbalancer geschreven worden in C#. Deze loadbalancer moet op TCP-niveau communiceren met de gekoppelde servers. De communicatie die plaatst vind moet gebeuren als een client naar de webpagina van de loadbalancer navigeert.

Deze loadbalancer moet verschillende algoritmes kunnen gebruiken om een geschikte server te kiezen. Deze algoritmes moeten eenvoudig worden uitgebreid. De drie standaard meegeleverde algoritmes zijn; Round Robin, Random en Load. Daarnaast moet er in de instellingen van de loadbalancer cookie en session persistentie ingeschakeld kunnen worden. Als een van deze persistentie wordt ingeschakeld worden de algoritmes genegeerd en zal de juiste server gekozen moeten worden op basis van cookie of server.

In de afgelopen week heb ik zo een loadbalancer gebouwd. In dit document zal ik reflecteren op mijn ontwerpkeuzes en de beroepsproduct zelf.

In het hoofdstuk “Ontwerpkeuzes” is per paragraaf elk functionaliteit van de load balancer beschreven en de gemaakte ontwerpkeuzes toegelicht. In de volgende hoofdstuk is de reflectie op de complete beroepsproduct te vinden. Hierin is toelichting op de architectuur, design en implementatie te vinden. Daarnaast is er dit hoofdstuk een aantal verbeter punten voor de load balancer.

# Ontwerpkeuzes

Tijdens het bouwen van de load balancer heb ik bepaalde keuzes gemaakt. In dit hoofdstuk zal ik deze keuzes per onderdeel gaan benoemen en toelichten.

## Load verdeling

De load balancer moet in staat zijn om door middel van een algoritme een server de kiezen die de client verzoek afhandelt. De huidige load balancer ondersteund standaard de volgende drie algoritmes:

* Random
  + De load balancer selecteert willekeurig een beschikbare server
* Round Robin
  + De load balancer selecteert de beschikbare servers één voor één, totdat die bij de laatste server is. Hierna begint de load balancer weer bij de eerste server en herhaalt het proces zich.
* Load
  + De load balancer selecteer de beschikbare server die de minste verzoeken heeft behandeld.

### 2.1.2. Codevoorbeeld

Reflection (BalanceStrategyService)

public ObservableCollection<IStrategy> GetLoadStrategies()

{

string[] files = Directory.GetFiles(@".\", "\*.dll"); // Defines the path to look for DLL files

foreach (var file in files)

{

Assembly assembly = Assembly.LoadFrom(file);

foreach (Type type in assembly.GetTypes()) // Gets the list of types, the parents are also included in this list

{

if (type.GetInterfaces().Contains(typeof(IStrategy))) // Checks if the type is a type of IStrategy, the balance interface

{

IStrategy balanceStrategy = Activator.CreateInstance(type) as IStrategy; // Creates a instance of the object as an IStrategy object.

\_loadBalanceStrategies.Add(balanceStrategy); // Adds the strategy to the list of strategies.

}

}

}

return \_loadBalanceStrategies;

}

Random

public Server GetBalancedServer(ObservableCollection<Server> servers)

{

Random rnd = new Random();

var serversAlive = servers.Where(server => server.isAlive == true);

return serversAlive.Skip(rnd.Next(serversAlive.Count())).FirstOrDefault(); ;

}

Round Robin

public Server GetBalancedServer(ObservableCollection<Server> servers)

{

List<Server> serversAlive = servers.Where(server => server.isAlive == true).ToList();

if (Position >= serversAlive.Count)

Position = 0; // Position is a property of the RoundRobinAlgortihmStrategy

Server server = serversAlive[Position++];

return server;

}

Load

public Server GetBalancedServer(ObservableCollection<Server> servers)

{

List<Server> serversAlive = servers.Where(server => server.isAlive == true).ToList();

int lowestRequest = -1;

Server serverWithLowestLoad = null;

foreach (Server server in serversAlive)

{

if (server.RequestHandledCount < lowestRequest || lowestRequest == -1)

{

lowestRequest = server.RequestHandledCount;

serverWithLowestLoad = server;

}

}

return serverWithLowestLoad;

}

### 2.1.3. Alternatieven

### 2.1.4. Bronnen

## Persistentie

Een client kan cookies meesturen in zijn HTTP-verzoek. De load balancer kan er voor kiezen om de client een cookie mee te geven, zodat deze bij de eerst volgende verzoek naar dezelfde server wordt geleid. Naast een standaard cookie kan de server een session naar de client versturen.

Als de cookie persistentie is ingeschakeld, dan moet de load balancer controleren of er in het verzoek een cookie is meegestuurd dat de adres van de server bevat. Als die er is dan moet de load balancer deze server selecteren en de verzoek laten behandelen. Is dit niet het geval dan wordt er willekeurig een beschikbare server gekozen en wordt er bij de client een cookie opgeslagen met de verwijzing naar de server.

Een session wordt aangemaakt door de server. Deze houdt alle data van de client bij, zodat de client niet wordt overspoelt met cookies. De session bevat een ID en wordt verstuurd naar client. De client slaat deze sessionID op in zijn cookies. Bij session persistentie mag de client niet weten met welke server die gekoppeld is. De load balancer is dan verantwoordelijk om de sessionID te koppelen met de juiste server. Bij het ontbreken van een sessionID wordt een willekeurig server gekozen.

### 2.2.2. Codevoorbeeld

### 2.2.3. Alternatieven

### 2.2.4. Bronnen

## Health monitoring

Een server kan vastlopen of zelfs uitgeschakeld zijn. De load balancer moet regelmatig controleren of een server nog verzoeken kan behandelen. Als een server geen verzoeken meer kan behandelen, dan mag de load balancer hier geen verzoeken naar sturen. Als een client een sessie of cookie heeft moet de load balancer deze client laten zien dat de server niet beschikbaar is. Hiervoor wordt er gebruik gemaakt van de HTTP-code 503.

### 2.3.2. Codevoorbeeld

### 2.3.3. Alternatieven

### 2.3.4. Bronnen

# Beroepsproduct

Tijdens het bouwen van dit beroepsproduct had ik een tunnelvisie over het gebruik van reflection. Maar dit veranderde al vrij snel toen ik mij hierin had verdiep. Als de load balancer uitgebreid moet worden terwijl die in productie is kan dit hier vrij eenvoudig mee. Tijdens het runnen kan je namelijk DLL’s toevoegen in de daarvoor bestemde folder. Hiervoor zou er nu wel een knop toegevoegd voor moeten worden.

Bovendien kan een extern programmeur zelf een balance strategy kunnen toevoegen. Hiervoor moet die zich alleen aan de strategy interface te gaan houden. De strategy moet deze interface gaan implementeren.

Waar ik persoonlijk erg trots op ben is het verspreiden van alle functionaliteiten voor de loadbalancer. Elk belangrijk onderdeel uit de proxy is hierin terug gekomen als een eigen class library. Zie hiervoor de het ontleden van de HTTP requets en het communiceren met een server en client. Deze class libraries kunnen nu herbruikt worden voor een ander project dat draait op .NET Core 3.1.

*Definieer kwaliteit in je architectuur, design, implementatie. Geef voorbeelden.*

*Wat kan er beter, waarom?*

Verbeteringen

De model LoadBalancerModel is nu groot geworden. Ik had dit graag nog opgesplitst willen hebben. De funcionaliteiten horen wel bij een typisch load balancer, maar hierin zou de LoadBalancerModel een controller functie kunnen hebben.

De health monitor controleert nu om de 2 seconde de servers.

# Conclusie