Architectuur voorstel

Abstract
Voorstel voor architectuur voor een applicatie welk stock voorspelling maakt adhv. vorige metingen en trends data. Sterke nadruk op onafhangkelijke services om monolitsche blob te vermijden.

Keywords
WAMP — RPC — Services — Stock

Contents

1. Overzicht

Een client kan een stock voorspelling/vergelijking uitvoeren door een Remote Procedure Call(RPC) op de back-end uit te voeren. Alle berekeningen gebeuren op de back-end. De back-end is een server applicatie welke een interface heeft naar twee data sources:

1. Google trends data (GTD)
2. Stock data (SD)

Voor zowel SD als GTD kan een onafhangkelijke service gemaakt worden welke de data ophaalt. Voor SD lijkt de Bloomberg API [blo()] interessant. Voor GTD lijkt NodeJS trends API interessant [tre()]. De gehele architectuur staat voorgesteld in figuur 1. In deze figuur is de state de service welke uiteindelijk de number crunching doet. Alle services zijn onderling geconnecteerd mbv. een WAMP [cro()] router. Een applicatie die ongeveer deze architectuur benadert is te vinden op een github repo [git()] van mij. Merk hierbij op dat de state al grotendeels geïmplementeerd is en er al gebruikt gemaakt wordt van NodeJS services.

2. Voordelen

Voordelen van het werken met WAMP en de verschillende services als data source:

1. Zolang verbinding met de WAMP router actief is, moet er niet gedacht worden aan connectie status etc. WAMP router impl verzorgt dit. Bestaande WAMP routers (bv Crossbar.io, gunstig gelicensed) zorgen ervoor dat de RPCs bij de juiste service terecht komt zonder veel moeite.
2. Gemakkelijk nieuwe services registreren bij de WAMP server zonder te veel hassle. Stel dat op termijn een andere data source er bij komt kan die gewoon bij de WAMP router geregistreerd worden en met weinig aanpassing in de state service geïntegreerd worden.
3. Omdat de clients doorheen de WAMP router naar de state service moeten kan ook makkelijk (indien nodig) een authenticatie service toegevoegd worden.
4. Volledige loskoppeling van de client waardoor gemakkelijk clients gemaakt/uitgebreid kunnen worden. De API is eenvoudig een WAMP RPC call uitvoeren. Zie als voorbeeld 'Autobahn'. Autobahn is een library voor Python, C++, JS, ... welke met de router verbinding maakt en RPCs uitvoert. Voor een voorbeeld hiervan zie de scripts subdirectory van STC [git()].
5. Client is eenvoudiger en lightweight; stealth patches mogelijk.

3. Nadelen

Nadelen van het werken met WAMP en de verschillende services als data source:

1. Inherent aan het systeem is toegevoegde lag op berekeningen doordat data geroute moet worden naar back-end en dan terug naar de client.
2. Beduidend meer man uur nodig op korte termijn om de applicatie gemaakt te krijgen.
3. Single point of failure; WAMP kapot alles kapot. State kapot alles kapot.

References

[blo()] Bloomberg api: Open market data initiative. URL `https://www.bloomberglabs.com/api`.

[cro()] Crossbar.io: Networking for apps. URL `http://www.crossbar.io`.

[git()] Yoko stc github repo. URL `https://github.com/Yoko-Mao/STC/tree/master/src`.

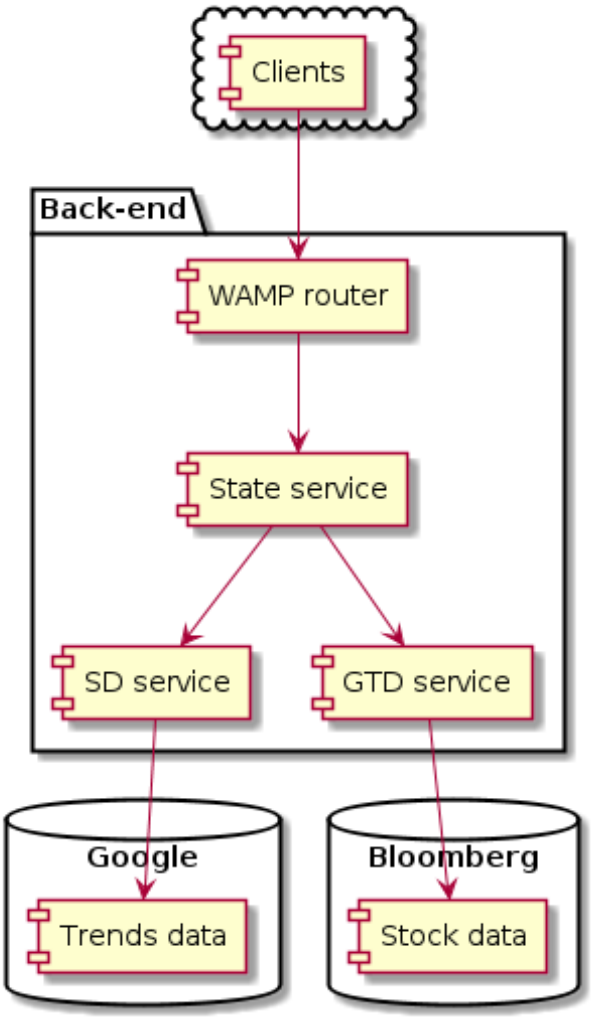

Figure 1. WAMP service architectuur

[tre()] An api layer on top of google trends. URL `https://www.npmjs.com/package/google-trends-api`.