Docker in de praktijk

Een kijkje achter de schermen bij Translink

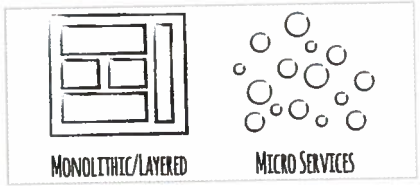
De afgelopen maanden zijn wij vanuit Info Support werkzaam geweest bij Translink. Dit is de organisatie achter de OV-chipkaart. Wekelijks verwerkt Translink 45 miljoen transacties en er zijn 14,4 miljoen actieve OV-chipkaarten (bron: translink.nl).

De wens van Translink was om het systeem achter de website (**www.ov-chipkaart.nl**) en de mobiele applicaties te vernieuwen. De speerpunten bij dit project waren een toekomst-vaste architectuur neerzetten, een sterk verkorte time-to-market realiseren en het oude CMS vervangen om zodoende web redactie in staat te stellen om makkelijker en sneller wijzigingen te kunnen doorvoeren. Verder waren er nog een tweetal belangrijke non-functional requirements: een piekbelasting van 20.000 bezoekers per uur op de website en maximaal 15 minuten per week downtime (in de nacht). Het doel van dit artikel is om de ervaringen te delen die wij tijdens dit leuke en uitdagende traject bij Translink hebben opgedaan met de inzet van Docker.

Van macro-service naar micro-service

In traditionele softwareontwikkeling kom je regelmatig applicaties tegen die bestaan uit honderdduizenden regels code. Changes zijn kostbaar, leiden tot regressie en worden via grote releases slechts enkele keren per jaar uitgerold. Builds kosten uren en worden alleen nog maar 's nachts uitgevoerd. Omdat productie maanden achter loopt op ontwikkeling zwerven codebases steeds verder uit elkaar en wordt de situatie al snel onoverzichtelijk. Een release is "zwaar", "risicovol" en "moet gemanaged worden".

Met het ov-chipkaart.nl vernieuwingstraject hebben wij ons gericht op de ontwikkeling van een systeem waar wij, in vol vertrouwen van de klant, wijzigingen binnen een uur in productie kunnen hebben staan. Om dit doel te bereiken hebben wij gekozen voor een architectuur op basis van micro-services (zie **figuur 1**).



Figuur 1

Zo hebben wij bij Translink aparte micro-services ontwikkeld voor:

- het opslaan en bijsnijden van foto's;
- het genereren van documenten;
- het verwerken van bestellingen;
- het autoriseren van gebruikers;
- het vertalen van postcodes naar adressen;
- het vragen naar kaart of klantgegevens;
- en nog veel meer...

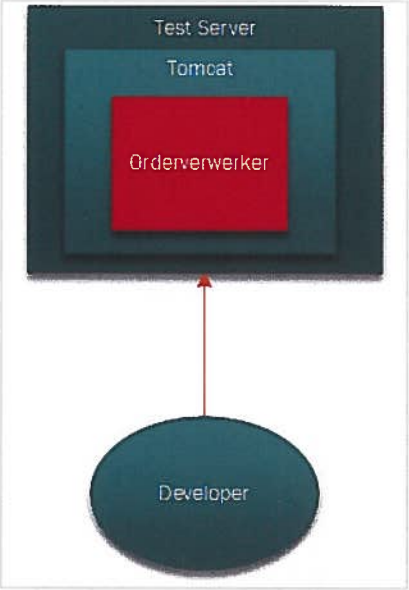
Dit heeft geleidt tot een systeem bestaande uit 16 op zichzelf staande componenten met 28 onderliggende relaties. Ieder component kunnen we releasen zonder andere componenten te raken. Er is dus niet sprake van 1 applicatie met 1 versie die draait op 1 omgeving, maar 16 applicaties met ieder zijn eigen versie en omgeving. Het totale systeem is hiermee organisch en evolueert naarmate individuele componenten worden geüpdatet.

Een systeem op basis van micro-services architectuur is van nature schaalbaar en robuust. Als bijvoorbeeld het foto-management component crasht, heeft dit alleen impact op de foto-uploadtool van de website. Reizigers die hun reistransacties bekijken, saldo bestellen of de app gebruiken, merken hier niets van. Mocht het foto-management component niet snel genoeg zijn, omdat bijvoorbeeld de gezichtsherkenning veel CPU-tijd kost, dan kunnen we eenvoudig meerdere instanties van de fotomanager online brengen. Dit wordt in de infrastructuur gerealiseerd door een spreiding van

Wouter van de Ridder is werkzaam als Software Engineer bij Info Support

Sjoerd van Oostenbrugge is allround software engineer bij Info Support

Martin Devillers is een software architect bij Info Support die organisaties graag helpt met het ontwerpen en ontwikkelen van krachtige doch eenvoudige IT-oplossingen



Figuur 2

onze docker containers over meerdere hosts waarbij het verkeer via een loadbalancer verdeeld wordt.

Waarom Docker?

Voor de realisatie van deze architectuur zochten we een oplossing om automatisch omgevingen uit te rollen over onze OTAP-straat. Dit om met veel vertrouwen snel te kunnen releasen. Daarnaast zocht Translink ook een toekomst-vaste architectuur. Docker helpt ons om een flexibele infrastructuur op te zetten die gemakkelijk uit te breiden is in de toekomst. Hoewel Docker zelf nog volop in ontwikkeling is, zijn de onderliggende technologieën in de Linux kernel al een aantal jaren stabiel.

Wat verder heeft geholpen bij het maken van de keuze voor Docker is het feit dat zowel het project als het beheer bij Info Support ligt en er gewerkt werd in een DevOps setting waardoor het een stuk eenvoudiger was om de benodigde kennis te borgen en draagvlak te creëren. Vanuit Info Support was er al eerder geïnvesteerd in Docker kennis, hetgeen ons heeft geholpen om het project op te starten.

Hoe zijn wij ermee gestart?

Om een snelle start te kunnen maken met Docker hebben we een aantal ervaren Info Support collega's gevraagd om ons te begeleiden. Allereerst hebben ze een presentatie aan de projectleden uit de DevOps teams gegeven om de basiskennis over Docker op te schroeven. Hierna zijn wij in een kleine groep verder gegaan. We hebben een relatief simpel component gepakt als startpunt om te Dockerizen.

Inrichting Continuous Deployment

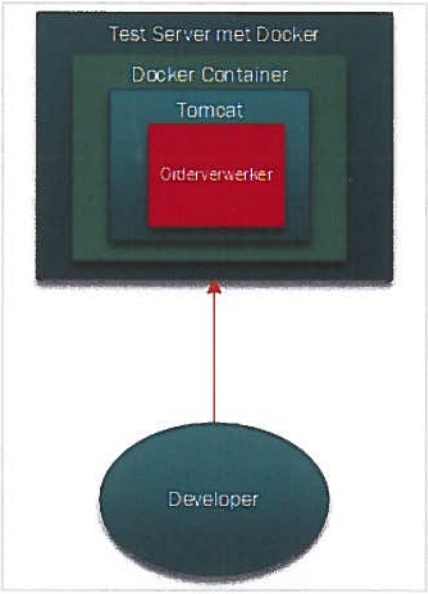
Voor de realisatie van een korte time-to-market hebben we een continuous deployment straat ingericht waarin we ieder component met één druk op de knop kunnen releasen.

Simpel begin

Voor de inrichting van ons continuous deployment proces zijn we eenvoudig begonnen en hebben we gedurende het project het proces stapje voor stapje uitgebreid. We zijn gestart met een handmatige installatie van Tomcat, direct op de Linux machine. (**figuur 2**) Deze keuze is bewust gemaakt zodat we konden ervaren wat we allemaal nodig hadden in een traditionele omgeving, waarbij we niet direct met Docker-specifieke uitdagingen te maken kregen.

Docker

Hierna is Docker geïnstalleerd op deze machine (**figuur 3**). Vervolgens hebben we gekozen om te beginnen met het officiële Tomcat Docker image als basis voor onze eigen image, welke op zijn beurt weer gebaseerd is op Ubuntu en de OpenJDK. Officiële images staan aangegeven in de Docker Hub als 'Official repository' en het gebruik hiervan wordt aanbevolen door Docker. De officiële images worden gepubliceerd en gecontroleerd door een toegewijd



Figuur 3

team bij Docker en worden ook tijdig voorzien van security updates. Deze image is uiteindelijk zo goed bevallen dat we die in productie nog steeds gebruiken. In **figuur 3** staat een verkorte en geanonimiseerde versie van één van onze Dockerfiles.

Op het moment dat deze container gestart wordt, zal Tomcat opstarten en de toegevoegde WAR file deployen. Het start.sh script geeft ons de mogelijkheid om voor het starten van Tomcat eventueel nog andere acties uit te voeren, in dit geval start het echter gewoon de catalina.sh van Tomcat.

Images ontwikkelen

Omdat vrijwel alle ontwikkelaars bij ons werken op Windows of OS X machines was het een uitdaging om Docker images snel te testen. Een kort onderzoek leidde ons naar Boot2Docker, maar zowel op Windows als op OS X zaten hier teveel haken en ogen aan. Eén van de krachten van Docker is dat je juist met weinig inspanning in korte tijd een werkende omgeving hebt staan. Als je toch in Windows of OS X wilt werken, raden wij aan om met een virtuele machine op basis van Ubuntu te werken. In ons geval hebben wij ervoor gekozen om een centrale Docker server te installeren, specifiek om de Docker images op te ontwikkelen.

Nadat we met wat experimenteren op deze server ervaring hadden opgedaan met Docker images bouwen, zijn we aan de slag gegaan met het inrichten van een POM file om onze Docker images te kunnen bouwen via Maven. Deze inrichting stelt ons in staat om in onze Maven Docker POM bijvoorbeeld een dependency op te nemen naar onze eigen WAR file.

Configuratie

Daarnaast hebben we ook een splitsing gemaakt tussen de applicatie en de omgevingsafhankelijke configuratie. We hebben

```
FROM tomcat:8-jre8
MAINTAINER Info Support

# Expose tomcat port
EXPOSE 8080

# Add the WAR file
ADD orderverwerker.war /usr/local/tomcat/webapps/orderverwerker.war
# Add startup script to the container
ADD start.sh /usr/local/

# Launch Tomcat on startup
CMD ["sh", "/usr/local/start.sh"]
```

Listing 1

gekozen om hierbij gebruik te maken van 'data volume containers'. Concreet zijn dat Docker containers die alleen een Docker volume bevatten en geen draaiend proces. Om dit te realiseren, hebben we een Dockerfile geschreven waarbij we aan de hand van ADD statements de benodigde configuratie bestanden hebben toegevoegd aan dit image. Vervolgens hebben we de directory met configuratie bestanden beschikbaar gesteld als Docker volume.

Listing 2 is een voorbeeld van een Docker volume container.

Bij het opstarten van de tomcat applicatie container verwijzen we naar het aangemaakte volume met de '--volumes-from' optie van het Docker run command (zie **Listing 3**).

Een groot voordeel van deze opzet is dat de configuratie netjes gescheiden is van de applicatie, zoals te zien is in **figuur 4**. Dit staat ook toe dat de configuratie aangepast kan worden zonder dat het nodig is om een nieuwe versie van de applicatie uit te rollen
.

Automatisch deployen

De volgende stap in de inrichting van ons continuous deployment proces was het inrichten van Jenkins build jobs. Het einddoel was om in Jenkins één pipeline per Docker container te krijgen waarmee we automa-

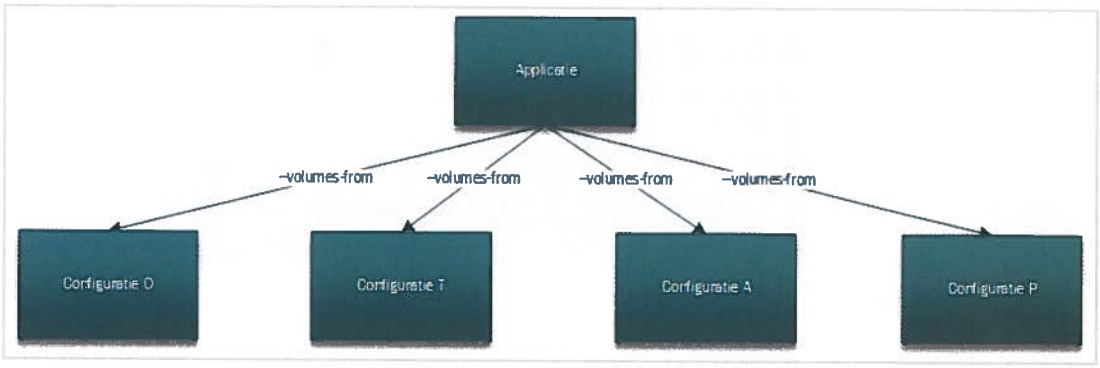
```
FROM ubuntu:14.04.1
MAINTAINER Info Support

RUN mkdir -p /var/config
VOLUME /var/config

ADD orderverwerker.properties /var/config/orderverwerker.properties

CMD ["/bin/bash"]
```

Listing 2

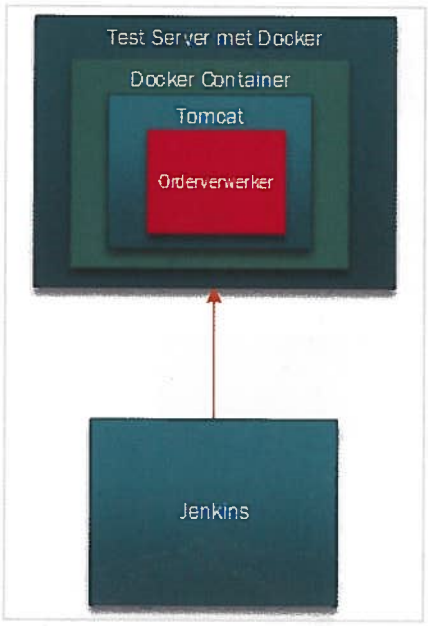
```
docker run -d --name orderverwerker --volumes-from orderverwerker-configuration
```

Listing 3

DANKZIJ DOCKER IS HET ERG EENVOUDIG OM NIEUWE APPLICATIES INCLUSIEF OMGEVING AAN JE APPLICATIE-LANDSCHAP TOE TE VOEGEN



Figuur 4



Figuur 5

tisch, of met een druk op de knop, kunnen uitrollen over de OTAP (**figuur 5**). De initiële job in Jenkins wordt getriggerd door een commit in Git. Jenkins checkt vervolgens de code uit, compileert deze, draait unit tests, voert sonar code analyses uit en upload de gebouwde WAR file naar Nexus (**figuur 6**, eerste blok).

De tweede job in de pipeline bouwt de Docker image. Met behulp van de eerder opgestelde POM was deze job eenvoudig in te richten. Aan het einde van de job uploaden we de gebouwde image als TAR file naar Nexus (**figuur 6**, tweede blok). We hebben hier gebruik gemaakt van Nexus in plaats van een Docker registry, vanwege de bestaande infrastructuur met betrekking tot tooling. Indien de mogelijkheid bestaat om binnen de infrastructuur van de organisatie een Docker registry op te zetten, dan heeft dit de voorkeur.

De derde stap in onze pipeline is het deployen van het gebouwde Docker image naar de eerste omgeving. Deze job bouwt als eerste het eerder genoemde Docker volume, omdat deze specifiek is per omgeving. Dan haalt Jenkins het eerder gebouwde applicatie image weer op vanuit Nexus en upload deze naar de server waar het moet komen te draaien via de Jenkins SSH plugin. Op deze server wordt de container gestart aan de hand van een startup script om een aantal Docker commands uit te voeren (**figuur 6**, derde blok).

Uitbreiding

Na deze inrichting zijn we aan de slag gegaan met het opzetten van een tweede Docker image (**figuur 7**). De reden hiervoor was dat we graag de onderlinge communicatie tussen Docker containers wilden testen, waarbij we inzichtelijk wilden maken hoe we omgingen met het exposen van poorten en of er een merkbare performance overhead was op netwerk niveau. Nadat de testen succesvol waren afgerond en geen problemen waren gevonden, zijn we verder gegaan met de volgende stap.

Figuur 6

Acceptatie & productie

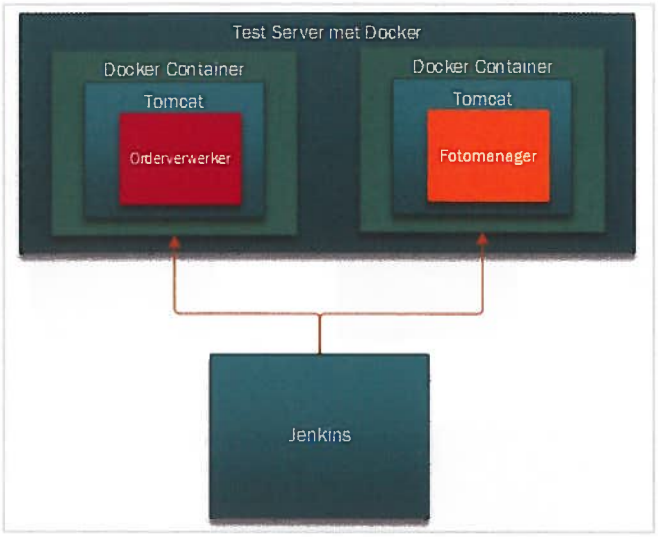
Nadat het tweede image ook op de testomgeving was ingericht, zijn we begonnen met het inrichten van de acceptatieomgeving. De grootste uitdaging hierbij was het feit dat deze omgeving in een ander datacenter staat dan onze Jenkins server. Uiteindelijk is de keuze gemaakt om in de acceptatie omgeving een Jenkins slave in te richten die onze deployments daar kon uitvoeren. Door gebruik te maken van een slave hebben we nog steeds toegang tot en beheer op onze builds vanuit dezelfde Jenkins master en kunnen we ze in dezelfde deployment pipeline weergeven en beheren. Verder zijn de acceptatie- en productieomgeving dubbel uitgevoerd om de hoge uptime te kunnen garanderen. Dankzij deze dubbele uitvoering kunnen we nu ook zonder downtime de deployments uitvoeren in de acceptatie- en productieomgeving.

Als laatste hebben we de productieomgeving op een vergelijkbare manier ingericht, waarmee we de deployment pipeline voor onze componenten konden afronden. De beschreven pipeline staat afgebeeld in **figuur 8**. De daadwerkelijke pipeline bij Translink is uitgebreider, want deze bevat ook nog andere stappen, zoals geautomatiseerde (integratie) tests en het stapsgewijs uitrollen van de containers over de verschillende datacentra om downtime te voorkomen.

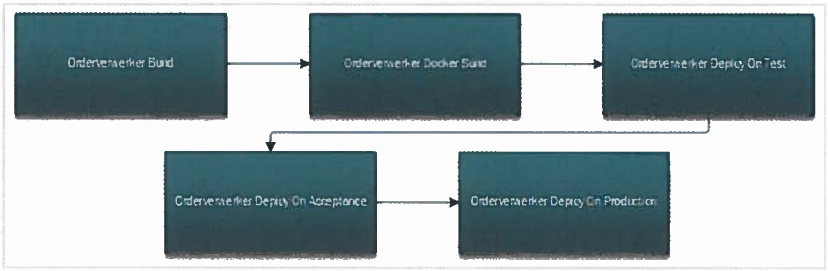
Tot Slot

We hebben de website en achterliggende frontoffice nu in productie draaien. Van zes uur 's ochtends tot twaalf uur 's nachts zitten honderden gebruikers tegelijkertijd op de website. Dit leidt tot miljoenen berichten die dagelijks tussen de micro-services worden uitgewisseld. Al deze micro-services draaien 24/7 stabiel binnen Docker containers en vereisen verrassend weinig resources.

Het hebben van 16 individuele applicaties klinkt als een nachtmerrie voor beheer, maar dankzij Docker is het juist erg eenvoudig om nieuwe applicaties inclusief omgeving aan je applicatielandschap toe te voegen. In traditionele situaties is het optuigen van een omgeving inclusief besturingssysteem, patches en configuratie een dagtaak voor beheerders. Met Docker is een omgeving praktisch 'wegwerpbaar'. Met enkele commando's definieer je een omgeving en breng je deze online. Bij een release installeer je geen software in een omgeving, maar vervang je de complete omgeving door een nieuwe. En dit alles in enkele seconden!



Figuur 7



Figuur 8

De investering die we gedaan hebben in het inrichten van de deployment pipeline en het gebruik van Docker heeft zich zeker terugbetaald. We bouwen nog steeds de containers op 0 en we kunnen de Docker componenten meerdere keren per dag zonder downtime naar productie deployen dankzij de dubbele uitvoering. Daarnaast kunnen we ook toekomstige Java of Tomcat updates meenemen over onze OTAP, waardoor we dit zonder handmatige installaties met vertrouwen in productie kunnen brengen. De inzet van Docker in dit project is een groot succes geweest en gezamenlijk hebben we een toekomstvast en flexibel landschap neergezet. ■

DE INZET VAN DOCKER IN DIT PROJECT IS EEN GROOT SUCCES GEWEEST