Enterprise

GlassFish is een Open Source Java EE project dat standaardoplossingen biedt voor een uitgebreid pakket van componenten die door de Java EE-standaard worden gedefinieerd. GlassFish bestaat inmiddels drie jaar. Veel bedrijven die het gebruiken, zetten dit product in als alternatief voor onder andere Oracle SOA Suite, BEA Weblogic en IBM WebSphere. De reden daarvan is dat GlassFish goedkoper is in aanschaf, betere en snellere implementatie kent en zeer stabiel is. Tijd dus voor een nadere kennismaking.

Mooie toekomst voor GlassFish

De ideale open source SOA enterprise server

Joost Hofman
zijn senior J2EE/SOA consultants bij Yenlo.

Thijs Volders
zijn senior J2EE/SOA consultants bij Yenlo.

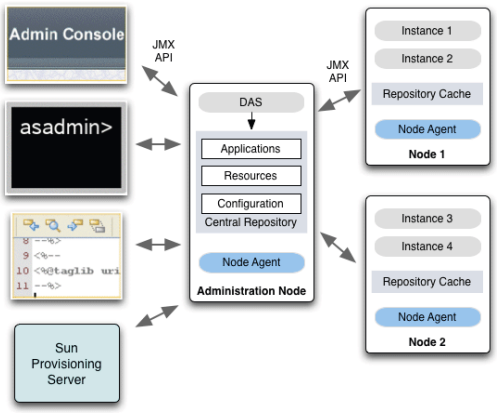
GlassFish bestaat uit twee stromen: GlassFish Applicatie Server (AS) en de GlassFish Enterprise Service Bus (ESB). De GlassFish AS is een high-performance Open Source applicatie server die Java EE 5 implementeert.

De mogelijkheden van GlassFish zijn te uitgebreid om in één artikel te behandelen. Daarom beperken we ons tot een aantal onderdele. In een vervolgartikel laten we meer componenten de revue passeren. Tot slot geven we een kort overzicht van de te verwachten eigenschappen van GlassFish V3 en OpenESB.next.

Sterk punt

Clustering is één van de sterkste punten in GlassFish. Clustering van applicatieservers verbetert namelijk de schaalbaarheid en beschikbaarheid en is ongelofelijk makkelijk te configureren.
Een cluster heeft altijd een Domain Admin Server (DAS) en één of meerdere nodeagents. Op de nodeagents kunnen meerdere applicatie server instanties aanwezig zijn. De DAS is het centrale punt van administratie en beheert de configuratie van de cluster. De DAS kan op vele manieren beheerd worden; admin console, command-line,

IDE zoals Netbeans of Eclipse en via de Sun XVM Ops Center.
De nodeagent is een lichtgewicht instantie van de AS. Op elke server is één nodeagent aanwezig. Een nodeagent regelt de levenscyclus van de AS-instanties. Dit betekent dat de nodeagent de



Afbeelding 1: Clustering Overview

instanties aanmaakt, start, stopt, verwijdert en indien de instantie als proces niet meer aanwezig is deze ook opnieuw start.
Dit zijn alle losse componenten binnen een cluster. Maar hoe communiceren deze met elkaar? De communicatie tussen de DAS en NA gaat via JMX en HTTP. De DAS communiceert met de nodeagents wanneer een configuratie verandert en wanneer gecontroleerd wordt of de nodeagent nog online is.
De NA communiceert bij het opstarten, of als deze aangemaakt wordt, met de DAS voor een synchronisatie van hemzelf of de instanties onder hem.

Een cluster is heel simpel te installeren:

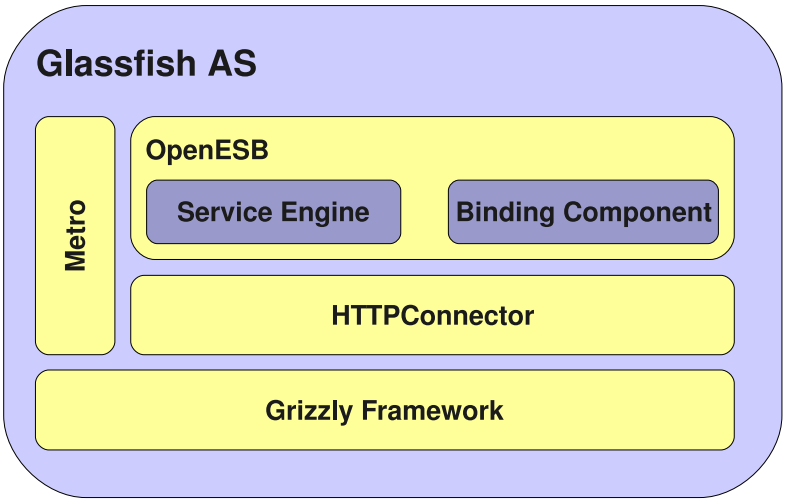
1. In de admin console kan clustering support aangezet worden met twee klikken.

2. Wanneer de GlassFish AS in cluster-mode draait, kan via bijvoorbeeld command-line een nodeagent aangemaakt worden. Dit gaat via de asadmin create-node-agent command.
3. Hierna kan de nodeagent gestart worden met asadmin start-node-agent.
4. In de admin console kunnen instanties op de nodeagents gemaakt worden en de loadbalancing ingesteld worden.

Bouwstenen

Nu het cluster geïnstalleerd is, kunnen we dieper ingaan op de belangrijke bouwstenen van GlassFish.

Grizzly is een multi-protocol (HTTP, UDP) framework gebaseerd op de Java NIO API. De Java NIO API staat voor Java New IO. Op dit moment kan deze API echter niet meer nieuw genoemd worden; hij werd al geïntroduceerd in Java SE 1.4. NIO levert (ten opzicht van de traditionele IO API) onder andere verbeterd buffer management, file I/O en regular expression ondersteuning. Sun positioneert de API als uitbreiding op de traditionele IO API. Eén van de belangrijkste features van NIO is Non-Blocking IO ondersteuning.



Afbeelding 2: Bouwstenen van GlassFish die hier behandeld worden

Traditionele Servlet-engines hanteren een Thread-per-request model. In principe bruikbaar in veel situaties, echter met de komst van Comet is dit model niet meer toereikend. Een Comet-client maakt een verbinding met de applicatieserver en wacht op events zoals een chatbericht of een nieuwe email. Als iedere Comet-client een thread toegekend zou krijgen, dan kan dit veel resourcegebruik tot gevolg hebben.

Grizzly maakt het mogelijk om sockets en threads te ontkoppelen. Zonder de NIO APIs is het niet mogelijk om een thread en socket van elkaar te ontkoppelen. De thread wordt geblocked, terwijl er gewacht wordt op binnenkomende gegevens. Grizzly regelt dat - wanneer er geen data beschikbaar is op een socket - de task waar Grizzly mee werkt en waar de socket in wordt vastgehouden, toegevoegd wordt aan een scheduler. Deze controleert periodiek of er data beschikbaar is op de socket. Als er data beschikbaar is, dan wordt een thread uit de threadpool gehaald en de task wordt binnen deze thread verder uitgevoerd.

Krachtige features binnen Grizzly zijn de ceiling en quota-instellingen. De ceiling-instelling kan ervoor zorgen dat er een maximum wordt gesteld aan het percentage threads dat aan een bepaalde applicatie mag worden toegekend. Een Quota-instelling zorgt ervoor dat er een bepaalde resource reservering wordt gedaan op de threadcapaciteit van de applicatieserver voor een bepaalde applicatie. Feitelijk betekent dit dat er een Service Level Agreement in Grizzly kan worden geconfigureerd en betere capaciteitsplanning kan worden gemaakt voor de applicatieservers.

Grizzly is evenals GlassFish ontworpen op basis van een pluggable architectuur. Veel belangrijke onderdelen van Grizzly kunnen met eigen implementaties van interfaces worden uitgebreid en aangepast. Onder andere de ThreadPool-implementatie, die door Grizzly wordt gebruikt is instelbaar, en Algoritmes welke de requestbytes inlezen/behandelen kunnen ingesteld worden enzovoort. Tevens kan de manier waarop Grizzly met IO omgaat worden ingesteld in traditionele IO of NIO.

Binnen GlassFish wordt Grizzly gebruikt als onderliggend framework voor de implementatie van de HTTP Connector van de applicatieserver. GlassFish ORB en GlassFish MQ maken gebruik van Grizzly voor low-level communicatie. De HTTP Bindingcomponent maakt gebruikt van Grizzly.

Grizzly is in ontwikkeling sinds 2005 en versie 1.9.0 is in december 2008 uitgekomen.

Metro is een webservice framework dat ontwikkelaars in staat stelt om webservice oplossingen te maken. Sun noemt het ook wel de one-stop shop voor webservice ontwikkeling. Het hoofdcomponent van Metro is de JAX-WS reference implementatie. Versie 1.2 van Metro ondersteunt onder andere JAX-WS 2.0 en WS-I Basic, Attachment en Simple SOAP Binding Profile 1.0. Metro ondersteunt ook WSIT (Webservice Interoperability Technology) die .NET 3.0 webservice communicatie mogelijk maakt .

JAX-WS hanteert eenzelfde model als voor EJB 3.0 wordt gebruikt. Dit betekent dat het niet nodig is om classes te extenden van webservice baseclasses of om webservices interfaces te implementeren.. In plaats daarvan kunnen annotaties worden aangebracht om een POJO tot webservice te transformeren. Veel van de XML die nodig is voor descriptors voor de applicatieserver wordt door annotaties verborgen waardoor JAX-WS makkelijk in gebruik is.

Het versturen van een grote hoeveelheid binaire data naar een webservice is mogelijk door de data met Base64 encoding in de SOAP enveloppe te plaatsen. Het encoden en decoden van deze data is een zwaar proces en kan er theoretisch voor zorgen dat de XMLparser zich 'verslikt' in de hoeveelheid data waardoor er een OutOfMemoryError optreedt. Een betere oplossing is om deze data via een SOAP attachment te versturen. Hierdoor hoeft de data niet encoded en decoded te worden en kan de server deze data direct verwerken.
Metro heeft DataHandler implementaties die ervoor zorgen dat de binnenkomende data naar een temporary file wordt weggeschreven om een OutOfMemoryError te voorkomen..
Schema validation is ook één van de features van Metro. Veel webservice stacks hebben schema validation default uit staan, omdat dit redelijke veel processorkracht vergt, zo ook Metro. Het is echter eenvoudig om schema validation weer aan te zetten. Een @Schemavalidation annotatie op de webservice class is voldoende om server-side validation aan te zetten. Client-side validation kan worden aangezet door bij het opvragen van de BindingPort een SchemaValidationFeature instance mee te geven.

Voorbeeld:
De laatste feature die we hier behandelen is de 'Start from Java' feature. Met behulp van Metro is het mogelijk om een webservice te testen door deze te starten in een normale Java runtime.
De webservice class kan worden uitgebreid met een main() method waarin de static method Endpoint.publish(.....) wordt aangeroepen. Als deze class gecompileerd wordt en door Wsgen wordt gehaald, dan is het mogelijk om vervolgens de gecompileerde javaclass te runnen in een JVM. Een lichtgewicht HTTP-server wordt gestart en de webservice wordt op het opgegeven endpoint beschikbaar gesteld. Deze feature kan vooral handig zijn bij het maken van unittest voor webservices.

```
proxy = service.getAddPort(new
SchemaValidationFeature());
try {
  proxy.add(12345, 20);
} catch(WebServiceException we) {
  // faalt de client-side validatie, request wordt
niet verstuurd naar de server
}
```

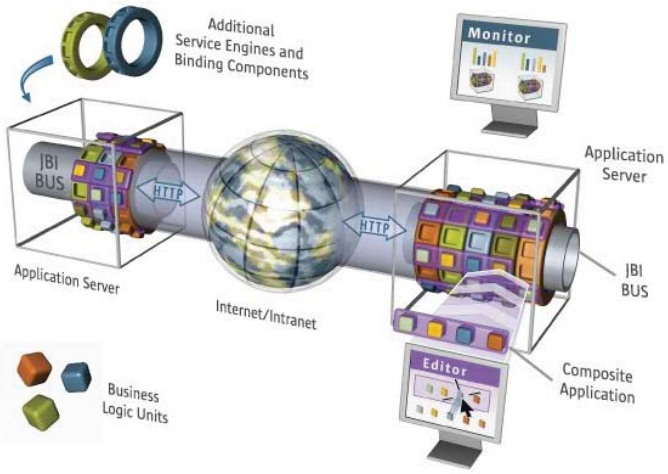
Open ESB

Een prominent onderdeel van de GlassFish technology-stack is OpenESB. Dit onderdeel is een open source implementatie van een Enterprise Service Bus.
GlassFish kent twee varianten van een ESB, te weten de GlassFish ESB en OpenESB. GlassFish ESB is een commerciële variant van OpenESB waar support op geboden wordt. OpenESB is open source en krijgt bijna dagelijks verbeteringen.
Wat is OpenESB? OpenESB is een op Java Business Integration (JBI, JSR-208) gebaseerde ESB implementatie. De basisingrediënten van JBI zijn een pluggable architectuur, volledig uit componenten opgebouwd. De componenten communiceren message driven oftewel gebaseerd op de WSDL's van de componenten. Implementaties van deze componenten zijn Service Engines (SE) en Binding Components (BC). De communicatie tussen deze SE's en BC's vindt plaats over een Normalized Messaging Router. Allemaal termen die door de JBI standaard worden gedefinieerd.
OpenESB heeft een aantal optimalisaties die ervoor zorgen dat berichten intern efficiënt doorgegeven kunnen worden. Alle berichten tussen componenten op dezelfde server instance worden in-memory doorgegeven waarbij gebruik gemaakt wordt van MTOM. MTOM is een SOAP-uitbreiding (SOAP Message Transmission Optimization Mechanism) die mogelijkheden biedt om het versturen van binary content in een SOAP-bericht te optimaliseren. Onder andere bespaart MTOM het encoden en decoden van base64 encoded binary attachments.
Service Engins bevatten business- en transformatielogica en kunnen communiceren met andere SE's. Binding Components zijn de bridges die de mogelijkheid bieden voor de Service Engines om te communiceren met objecten buiten de JBI bus zoals bijvoorbeeld databases of mailservers.

In figuur 3 is een high-level afbeelding van de OpenESB architectuur zichtbaar.

Links geïllustreerd is de JBI bus waar in SE's en BC's verschillende services geplugged zijn.
Rechts is eenzelfde JBI bus te zien waar dit eveneens getoond wordt. Hier wordt echter ook getoond dat met bijvoorbeeld een BPEL editor de componenten (services) in de JBI bus kunnen

Service Engins bevatten business- en transformatie-logica en kunnen communiceren met andere SE's

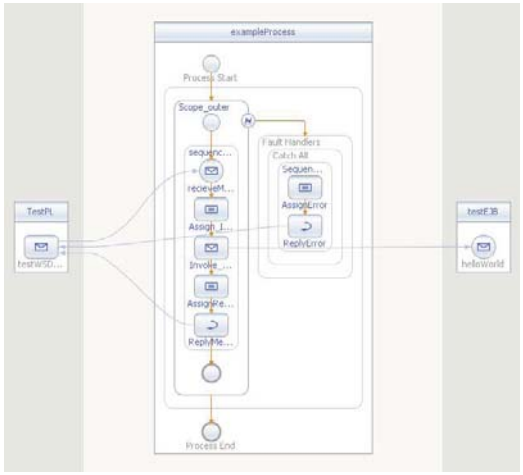


Afbeelding 3: OpenESB high-level architectuur

worden gekoppeld/georchestreerd tot een composite application. Tevens is een monitor afgebeeld waarmee de monitoring API wordt aangeduid. Later hierover meer.
Vanuit de OpenESB community zijn al vele SE's en BC's beschikbaar. Bijvoorbeeld BC's om te communiceren met een SMTP- en FTP-server en databases. Drie service engines die beschikbaar zijn binnen openESB worden nader uitgelicht.

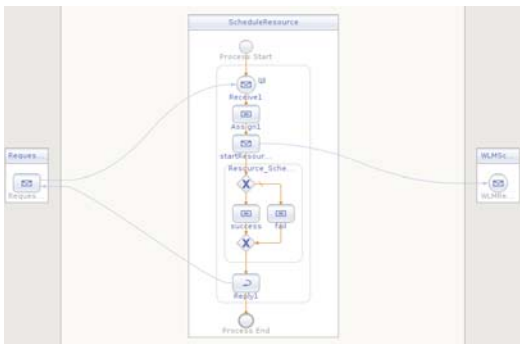
Java EE Service Engine zorgt voor het beschikbaar stellen van Java EE webservices binnen de JBI bus en voor het beschikbaar stellen van JBI componenten binnen de JBI bus aan deze Java EE webservices. Zonder de Java EE Service Engine zijn de Java EE componenten alleen via SOAP, HTTP of JMS benaderbaar en kan dus geen gebruik gemaakt worden van onder andere de NMR.
Om een JAVA EE component (EAR, WAR, JAR) als JBI component beschikbaar te stellen is het nodig dat een service-assembly wordt gemaakt waar dit Java EE component aan toegevoegd is (eventueel met meerdere andere service units) en een jbi.xml op te nemen voor dit Java EE component. Als de service-assembly deployed wordt, dan zal de applicatieserver een message versturen aan de Java EE Engine waarna deze de Java EE component beschikbaar stelt aan de JBI bus. In andere beschikbare ESB-producten als bijvoorbeeld Apache ServiceMix of Oracle ESB moet meer moeite gedaan worden om een EJB als webservice in de ESB op te nemen.
In Oracle ESB bijvoorbeeld moeten onder andere WSIF bindings worden aangemaakt om de EJB in de ESB beschikbaar te stellen. Transactiemanagement van de JBI bus naar de SE is ook geregeld en wordt doorgegeven van de JBI-bus naar de SE. De EJB of servlet die wordt aangeroepen zal in dezelfde transactie draaien als de rest van de services die binnen de composite application wordt gebruikt.
Een BPEL Service Engine is ook beschikbaar in GlassFish ESB. Zoals de naam verraadt maakt deze JSR 208-compliant JBI Service Engine het mogelijk om een BPEL proces te deployen in de ESB. De BPEL engine is WS-BPEL 2.0 compliant. Een BPEL proces is een goed voorbeeld van een composite application. Met behulp van een BPEL proces kunnen meerdere services worden gebruikt om tot een bepaald eindresultaat te komen.

Eerder is gesproken over clustering van de GlassFish Applicatie Server. De JBI runtime is standaard niet enabled om te draaien in een cluster. Het is echter wel mogelijk om meerdere BPEL Service engines binnen een GlassFish cluster te configureren alsof ze deelnemen aan een cluster. Het voordeel hiervan is dat fail-over en berichten correlatie beschikbaar komen voor de BPEL SE's in het cluster.



Afbeelding 4: Een voorbeeld BPEL proces

De BPEL SE kan gemonitored worden middels de Monitoring API. Deze API maakt het mogelijk om legio informatie over de BPEL processen in de service engine op te vragen. Ook is het mogelijk om de waarden van variabele in het BPEL proces aan te passen of om het BPEL proces tijdelijk stil te zetten.
Veelal kan een business proces niet zonder een vorm van interactie. Dit kan een menselijke interactie zijn maar ook een applicatie afhankelijkheid.
Dit is waar de worklist manager service engine gebruikt kan worden. De worklist manager kan nauw samenwerken met de BPEL SE om tijdens een BPEL proces bepaalde taken uit te voeren.
De Worklist manager is een separaat component en heeft geen directe koppeling met de BPEL engine ondanks dat deze twee kunnen samenwerken.
Eén van de redenen om de BPEL-engine en de worklist manager separaat te implementeren komt voort uit de component based architectuur en de sterke drang om standaard-compliant te blijven. BPEL uitbreiden met worklist management functionaliteiten maakt de BPEL complexer en niet uitwisselbaar. De BPEL-standaarden voorzien niet in de functionaliteiten die door de worklist manager worden geboden.
Veel voorkomende taken in een bedrijfsproces zijn



Afbeelding 5: BPEL WorkList Manager invocatie

invoer en uitvoer van gegevens, bijvoorbeeld het aanvragen van deployment resources. Dit houdt in dat een projectmanager een resource aanvraagt waarbij bepaalde gegevens ingevoerd moeten worden. Een resourceplanner krijgt deze gegevens binnen en kan de planning controleren en aanpassen. Als dit is gebeurd dan kan de projectmanager een reactie van het systeem krijgen waarin de aanvraag wordt gehonoreerd of afgewezen.

Middels een standaard Invoke activiteit kan in de WM een task worden gestart. De Worklist manager wordt geleverd met een applicatie waarin gebruikers de openstaande taken kunnen bekijken en hierop reageren.

Toekomst

GlassFish v3 is de ideale Web 2.0 applicationserver. Op dit moment is de prelude versie uit. Hierin zitten bijvoorbeeld al :

- Metro 1.4Java EE 6 ondersteuning
- OSGi ondersteuning
- JRuby (ook runtime) en Groovy support-Improved GlassFish update center waar bijvoorbeeld afhankelijkheden in packages naar voren komt, etc. Gebruik embedded GlassFish v3 in java applicaties
- Gaat nog beter met memory en disk space om
- Een scriptable command-line interface
- Automatische redeployment wanneer een file in een netbeans 6.5 project opgeslagen wordt

Project Fuji, ook wel openESB.next genoemd, werkt aan versie 3 van OpenESB. OpenESB v3 is net als OpenESB v2 gebaseerd op JBI, maar is daarbij ook gebaseerd op OSGI. Omdat OpenESB v3 in een OSGi bundel wordt geleverd kan deze op elke OSGi container geïnstalleerd worden. Dit kan bijvoorbeeld GlassFish v3 zijn of Apache Felix. Alle huidige BC's en SE's kunnen ook op OpenESB v3 draaien, dit komt door de JBI micro-kernel.
We kunnen ook vele nieuwe features verwachten. Bijvoorbeeld webbased tooling waarbij grafisch een ESB project gemaakt kan worden, Interceptors en filters zijn aanwezig en JRuby services waarmee bijvoorbeeld validaties gedaan kunnen worden.

Conclusie

GlassFish is de ideale open source SOA enterprise server, omdat het volledig op open source standaarden gebaseerd is. GlassFish is gemaakt door de mensen die de Java EE 5 standaard hebben gedefinieerd.
Ook de toekomst van GlassFish ziet er rooskleurig uit. GlassFish V3 belooft krachtige verbeteringen en kortere developmenttijden voor ontwikkelaars. Sun verwacht dat GlassFish V3 en Fuji gereleased worden in het derde kwartaal van dit jaar. «

Referenties

http://www.glassfish.nl
https://glassfish.dev.java.net/
.NET 3.0 support using JAX-WS:
http://blogs.msdn.com/dotnetinterop/archive/2008/07/29/connecting-to-exchange-using-jax-ws-part-1.aspx