

Automatische testbed monitoring voor toekomstig internetonderzoek

**Masterproef voorgedragen tot het behalen van het diploma van
Master in de industriële wetenschappen: informatica**

Andreas DE LILLE

*Promotoren: Geert VAN HOOGENBEMT
Piet DEMEESTER*

*Begeleiders: Brecht VERMEULEN
Wim VAN DE MEERSSCHE
Thijs WALCARIUS
Wim VANDENBERGHE*

Abstract

Door de huidige verschuiving naar cloudgebaseerde technologieën zal het belang van netwerkprotocollen en van de beschikbaarheid van netwerken alleen maar toenemen. Om deze verschuiving vlot te laten verlopen is er meer onderzoek nodig naar netwerktechnologieën. Dit onderzoek kan beter verlopen als onderzoekers en onderzoekscentra beter samenwerken. Hiervoor is FIRE (Future Internet Research and Experimentation) opgestart. FIRE is een Europees programma voor het uitvoeren van onderzoek naar het internet en de toekomst ervan. Hierbij werden universiteiten aangemoedigd om testbeds te bouwen.

Fed4FIRE is het sluitstuk van dit programma, waarbij al deze testbeds een universele API implementeren: SFA. Hierdoor wordt het delen van testfaciliteiten gemakkelijker gemaakt. Om te controleren of de SFA juist geïmplementeerd is, is jFed ontworpen. jFed wordt periodiek gebruikt door beheerders om hun SFA-API implementatie te controleren.

Deze masterproef bouwt hier verder op door een automatische monitoringsservice te maken. Deze kijkt periodiek of de SFA-API implementatie niet kapot is door nieuwe ontwikkelingen. Hierbij wordt informatie verzameld die via een monitoringsAPI ter beschikking gesteld wordt. Deze API (application programming interface) vormt een stevige basis waarvan andere toepassingen gebruik kunnen maken.

Binnen deze masterproef wordt de monitoringsservice ook uitgebreid met een loadtester. De loadtester wordt gebruikt om loadtesten uit te voeren. Hierbij wordt gekeken hoe een testbed reageert op een bepaalde belasting.

Abstract

Due to the new cloud-based technologies, network protocols and network reachability are now more important than ever. To make this change quick and clean, we need more and more network research. FIRE (Future Internet Research and Experimentation) is an European project created to improve the network and internet experimentation. FIRE is the opportunity to jointly develop potentially disruptive innovations for the future of the internet. It is about collaborative research and sharing test facilities. Researchers working for FIRE will now work closer together, sharing ideas. FIRE is also used to share test facilities from all over the world, so researchers have access to many different test facilities. This is done in Fed4FIRE, a project within FIRE.

Fed4FIRE is a program to implement a single SFA-API on each testbed within FIRE. To check if this was done correctly, jFed was created. As an extension of jFed, a monitoringsservice was created. The problem here is that this service is limited. It is not possible to check new API call. Recieving the monitoringinformation is also a problem.

This thesis will try to solve the aforementioned problem by creating an automated testbed monitoring system. A monitoringAPI will then share this information. Doing so, will provide future tools easy access to this information. Futhermore the thesis will also provide a tool to perform loadtest to test performance of testbeds.

Lijst van afkortingen

AM	Aggregate Manager
API	application programming interface
FED4FIRE	Federation 4 FIRE
FFA	First Federation Architecture
FIRE	Future Internet Research and Experimentation
FLS	First Level Support
GENI	Global Environment for Network Innovations
MA	Management Authority
PHP	PHP: Hypertext Preprocessor
PSQL	PostgreSQL
RSpec	Resource Specification
SA	Slice Authority
SFA	Slice-based Federation Architecture
SFA	slice-based federation architecture
SLA	Service Level Agreements

Lijst van figuren

2.1	De onderzoeker (boven) is de klant, de testbeds (onderhouden door de onderste personen) is de service provider. Fed4FIRE voorziet de link tussen beide partijen.	6
2.2	Eigenaars bepalen het beleid van hun testbed.	7
2.3	Een slice (geel) bestaat uit een verzameling slivers (groen).	9
2.4	Een onderzoeker, of de tool die hij gebruikt stuurt eerst een request RSpec en krijgt vervolgens een manifest RSpec terug.	10
3.1	FLS testbed monitoring	12
3.2	Resultaten van de stitching test	13
3.3	Geschiedenis van stitching test tussen wall1 en wall2	13
3.4	FLS testbed monitoring	15
3.5	Werking van de FLS monitor	16
3.6	Principe geni datastore	18
3.7	Een collector kan verschillende delen data ophalen.	19
4.1	De samenhang van de verschillende projecten in de masterproef.	22
5.1	De structuur van de databank	29
5.2	De werking van de API	31
5.3	FLS monitor	34
5.4	Overzicht van resultaten van de login testen.	35
5.5	Geschiedenis van een login test.	35
5.6	Naast de console uitvoer zijn ook het originele resultaat in XML formaat en het overeenkomstige HTML formaat beschikbaar.	36
5.7	De werking van de monitoringsservice	37
5.8	De uitwerking van de loadtesten.	39
5.9	Overzicht van de verschillende slices.	41
5.10	Status van de slices.	42
5.11	Belasting van het testbed, percentages moeten verdubbeld worden.	42
5.12	Belasting van een testbed met 100 gebruikers	43
5.13	Weergave per test	44

Inhoudsopgave

Lijst van figuren	v
Woord vooraf	ix
Inleiding	x
1 Inleidend kader	1
1.1 FIRE	1
1.2 GENI	2
1.3 Fed4FIRE	2
1.4 jFed	3
1.5 Opdracht: Monitoring van testbeds	3
2 SFA-Architectuur	5
2.1 Doel	5
2.2 Entiteiten	7
2.2.1 Owners	7
2.2.2 Operators	8
2.2.3 Researchers	8
2.2.4 Identity providers	8
2.3 Opbouw van een testbed	8
2.3.1 Component	8
2.3.2 Aggregate	8
2.3.3 Aggregate manager	9
2.3.4 Sliver	9
2.3.5 Slice	9
2.4 Communicatie met testbed via RSpec	10
3 Vooronderzoek	11
3.1 FIRE Monitor	11
3.1.1 Componenten	11
3.1.2 Testen	14
3.1.3 Werking FLS monitor	14
3.2 Tekortkomingen de FIRE monitor bij aanvang	17

3.2.1	Bereikbaarheid van de gegevens	17
3.2.2	Structuur van de gegevens	17
3.3	Integratie met GENI monitor	18
3.4	Besluit	20
4	Ontwerp monitor	21
4.1	Structuur	21
4.2	De API	23
4.3	De databank	23
4.4	De monitoringsservice	23
4.5	De loadtester	24
4.6	De website	24
4.7	Het GENI monitoringframework	24
4.8	Toekomstige ontwikkelingen	26
5	Implementatie van de monitor	27
5.1	Databank	28
5.1.1	Definities	30
5.1.2	Instanties	30
5.1.3	Configuratiegegevens	30
5.1.4	Resultaten	30
5.2	Webservice / API	31
5.2.1	Fasen	32
5.2.2	Parsen aanvraag	32
5.2.3	Aanmaken query	33
5.2.4	Uitvoeren query	33
5.2.5	Samenstellen van objecten	34
5.2.6	Encoderen van objecten	34
5.3	Website	34
5.4	Service	36
5.5	Loadtest	38
5.5.1	Uitwerking	38
5.5.2	Voorbeeld van een stresstest	40
A	MonitoringAPI reference	45
A.1	About	45
A.2	Introduction	45
A.3	Functions	46
A.3.1	List	46
A.3.2	Last	48
A.3.3	TestDefinition	49
A.3.4	TestInstance	49
A.3.5	Testbed	50
A.3.6	User	50

A.3.7	Q	51
B	Onderhoud, beheer en gebruik	52
B.1	Over	52
B.2	Toevoegen van testen (testinstancies)	53
B.3	Toevoegen van types/definities (testdefinitions)	55
B.3.1	Nieuw intern type	55
B.3.2	Testdefinitie toevoegen in de databank	57
B.3.3	User toevoegen	58
B.4	Troubleshooting	59
B.4.1	Resultaten worden niet opgeslagen door de API	59
B.4.2	Het programma loopt vast tijdens het uitvoeren van een zelf aange- maakte testen.	59
	Bibliografie	60

Woord vooraf

Vooreerst gaat mijn oprechte dank uit naar mijn promotor Geert Van hoogenbemt. Ondanks zijn drukke agenda kon ik regelmatig rekenen op zijn raad en advies. Daarnaast wil ik hem ook nog bedanken voor het nalezen en bijsturen van de scriptie.

Mijn dank gaat ook uit naar alle begeleiders van iMinds, die mij in de juiste richting gestuurd hebben. Wim Van de Meerssche en Thijs Walcarius wil ik hierbij bedanken voor hun feedback en enthousiasme over mijn werk. Daarnaast wil ik ook Brecht Vermeulen en Wim Vandenberghe uitdrukkelijk bedanken voor de kans om mijn masterproef uit te voeren bij iMinds.

Daarnaast gaat mijn dank ook uit aan mijn vrienden en familie voor de steun. Hierbij wil ik vooral Cedric De Pauw evenals Patrick Van Staey bedanken voor het geven van woord- en zinsuggesties. Daarnaast zorgden ze beide ook voor een aangename, toffe en stimulerende werksfeer.

Andreas De Lille
Gent, juni 2014

Inleiding

Het gebruik van netwerken en het internet om computers en allehande randapparatuur te verbinden zal in de toekomst alleen maar stijgen. Het is dan ook van groot belang dat onderzoek op dit gebied vlot en correct verloopt en dat onderzoekers samenwerken om zo ideeën en nieuwe technologieën te delen. Daarnaast moeten er ook testfaciliteiten zijn om deze nieuwe technologieën te testen. FIRE (Future Internet Research and Experimentation) is een Europees onderzoeksprogramma dat zich op deze doelen richt.

Om de configuratie en de werking van de verschillende testbeds gelijk te maken, is de federation architectuur ontworpen. De invoering hiervan zit in het Fed4FIRE (Federation for FIRE) project. De federation architectuur die in deze masterproef behandeld wordt, is SFA 2.0 (Slice-federation-architecture). Hierbij vormen alle testbeds van FIRE een federatie. Daardoor hebben onderzoekers binnen FIRE toegang tot alle testbeds binnen FIRE.

Het beheer van al deze verschillende testbeds is geen sinecure. Om dit beheer te vereenvoudigen is er binnen IBCN (Internet Based Communication Networks and Services), een onderdeel van het onderzoekscentrum iMinds, een monitoringsservice gemaakt. Deze service controleert periodiek of de SFA-API nog werkt. Echter door zijn snelle ontwikkeling is de service niet voorzien op uitbreidingen.

FIRE werkt samen met een gelijkaardig project, GENI. GENI (Global Environment for Network Innovations) is een Amerikaans project met gelijkaardige doelstellingen als FIRE. GENI is ook bezig met de ontwikkeling van een monitoringssysteem dat echter meer de nadruk legt op het monitoren van experimenten. De samenwerking tussen beide projecten kan bevorderd worden als beide monitoringssystemen compatibel zijn.

De opdracht van deze masterproef bestaat uit 3 grote delen.

Het eerste deel is het maken van een API die monitoringsdata beschikbaar maakt voor andere applicaties. Het tweede deel is een monitoringsservice maken die testbeds controleert. Het derde en laatste deel is de monitoringsservice uitbereiden om loadtesten uit te voeren. Met deze loadtesten wordt bekeken welke lading een testbed kan afhandelen.

De scriptie is al volgt uitgewerkt. Hoofdstuk 1 Situeert de masterproef. Hier wordt ook kort de opdracht uitgelegd. In hoofdstuk 2 wordt de SFA-architectuur uitgelegd. Deze architectuur wordt gebruikt om de configuratie en besturing van testbeds over heel de wereld gelijk te maken.

Hoofdstuk 3 geeft meer uitleg over de bestaande FIRE en GENI monitor en geeft hierbij ook de probleemstelling aan. Hoofdstuk 4 geeft bespreekt het ontwerp van de masterproef. In hoofdstuk 5 wordt de implementatie van de verschillende onderdelen besproken.

Bijlage A bevat de referentie van de monitoringsAPI, geschreven in Engels. Bijlage B bevat informatie voor de systeembeheerder. Bijlages A en B zijn specifiek gericht aan de systeembeheerder en eindgebruikers.

Hoofdstuk 1

Inleidend kader

In dit hoofdstuk wordt het achterliggende kader van de masterproef geschetst. Daarna wordt de opdracht uitgewerkt. De opdracht kan opgedeeld worden in drie delen. Het eerste deel is het maken van een API om de monitoringsinformatie beschikbaar te maken. Het tweede deel is het maken van een service die de monitoring uitvoert. Het derde en laatste deel is het maken van een loadtester die kijkt hoe een testbed reageert op een belasting.

1.1 FIRE

Deze masterproef is een onderdeel van een groter Europees onderzoeksproject genaamd FIRE (Future Internet Research and Experimentation). FIRE is gericht op onderzoek naar toekomstige internet- en netwerktechnologieën. Door onderzoekscentra te laten samenwerken (FIRE, 2014), tracht FIRE het onderzoek vlotter te laten verlopen. FIRE moedigde in eerste instantie aan om testbeds op te zetten om onderzoek te doen naar toekomstige internet-technologieën. Hierbij kwamen 2 doelen naar voor.

Het eerste doel is de samenwerking tussen verschillende onderzoekscentra verbeteren. Onderzoekers binnen eenzelfde vakgebied komen vaak gelijkaardige problemen tegen. FIRE vermijdt dat men telkens het wiel opnieuw uitvindt, door deze onderzoekers gemakkelijker en meer te laten samenwerken. Hierdoor worden oplossingen en ideeën meer gedeeld, zodat de ontwikkeling sneller kan verlopen.

Het tweede doel is het delen van testfaciliteiten gemakkelijker maken. Door FIRE krijgt een onderzoeker van een onderzoekscentrum toegang tot testfaciliteiten van andere onderzoekscentra binnen FIRE. Testfaciliteit is een algemene term die duidt op zowel de hardware als de software die gebruikt wordt om testen te verrichten. Een testfaciliteit of testbed is een server of een verzameling servers waarop men experimenten laat lopen.

Zo kunnen er op een testbed bijvoorbeeld een server en een aantal cliënten gesimuleerd worden. Deze worden verbonden met een aantal tussenliggende routers. Vervolgens wordt een videostream opgestart. Op deze videostream kan men storing introduceren door pakketten te droppen. De storing zal ervoor zorgen dat het beeld aan de client-side hapert. Om dit probleem op te lossen kunnen er technieken ingebouwd worden aan client-side. Zo kan er overgeschakeld worden naar een lagere kwaliteit indien blijkt dat de beschikbare bandbreedte onvoldoende is. Testen van degelijke technieken verlopen door gebruikt te maken van testbeds.

1.2 GENI

FIRE reikt echter verder dan Europa alleen, zo zijn er ook overeenkomsten met onderzoeksprojecten buiten Europa. Een voorbeeld daarvan is GENI (Global Environment for Network Innovations). Geni is een Amerikaans onderzoeksproject gericht om testfaciliteiten te bundelen en beschikbaar te stellen aan onderzoekers (GENI, 2014a). Door de samenwerking tussen beide projecten is het mogelijk dat onderzoekers geassocieerd met FIRE ook gebruik maken van testbeds binnen het GENI project en omgekeerd.

1.3 Fed4FIRE

Het probleem dat zich hierbij stelt is dat elk testbed een eigen interne werking heeft. Onderzoekers hebben nu wel toegang tot andere testbeds, maar moeten voor elk testbed eerst de specifieke configuratie leren. Verschillende testbeds laten samenwerken op deze manier is geen sinecure. Om dit tweede doel van het FIRE project eenvoudiger te maken, heeft men de federation architectuur ingevoerd.

De invoering van deze architectuur, binnen FIRE is een onderdeel van het Fed4FIRE-project (Federation 4 FIRE). De federation architectuur die hier gebruikt wordt is SFA 2.0. De bedoeling is dat testbeds binnen FIRE een federatie vormen. Een eerste gevolg hiervan is dat alle onderzoekers, services en testbeds binnen FIRE vertrouwd worden. Hierdoor heeft een onderzoeker binnen FIRE toegang tot alle testbeds. Een tweede gevolg is dat de configuratie en interne werking van de testbeds gelijk zijn. SFA wordt in hoofdstuk 2 besproken.

1.4 jFed

jFed werd door iMinds ontwikkeld (iMinds, 2014b) en is een javatool die in eerste instantie de SFA-API implementatie controleert. Systeembeheerders kunnen jFed aanwenden om te controleren of hun API juist werkt. Daarnaast is jFed uitgebreid zodat het ook gebruikt kan worden voor onderzoekers om testbeds aan te sturen. iMinds is een onafhankelijk onderzoekscentrum dat opgericht werd door de Vlaamse overheid (iMinds, 2014c). iMinds is leider van het Fed4FIRE project (iMinds, 2014a).

Met behulp van jFed kunnen onderzoekers snel en eenvoudig netwerken simuleren en testen uitvoeren. Toch is er nog ruimte voor verbetering in jFed. Een van de voornaamste problemen is dat een onderzoeker niet weet of het testbed dat hij gebruikt betrouwbaar is. Bepalen of een vreemd gedrag in een experiment te wijten is aan eigen ontwikkelingen of aan het falen van een testbed, kan op deze manier zeer tijdrovend zijn.

1.5 Opdracht: Monitoring van testbeds

Om dit probleem op te lossen heeft iMinds een monitoringssysteem uitgebouwd (Vermeulen, 2014) (Walcarius *et al.*, 2014). Merk op dat monitoring op meerdere niveau's mogelijk is zoals o.a. testbed, experiment, ... De monitoring die hier besproken wordt, richt zich op het controleren van testbeds en kijkt of een testbed online is en hoeveel resources er beschikbaar zijn.

Dit monitoringssysteem werkt, maar is door de snelle ontwikkeling niet voorzien op uitbreidingen. Bovendien is de informatie die hiermee verzameld wordt niet eenvoudig beschikbaar. De masterproef lost dit probleem op door een nieuw monitoringssysteem uit te bouwen. Dit systeem moet voorzien zijn op uitbreidingen zoals nieuwe soorten testen.

GENI heeft zelf ook een gedistribueerd monitoringssysteem uitgebouwd (GENI, 2014b). Deze service maakt gebruik van datastores (GENI, 2014c). Een datastore houdt de monitoring informatie van een testbed of aggregate bij. Deze informatie wordt dan opgehaald door een collector. Naast het uitbouwen van een nieuw monitoringssysteem, moet ook onderzocht worden of het mogelijk is om het monitoringssysteem te integreren in dit groter monitoringsframework. De werking van de GENI monitor wordt later in de scriptie uitgebreid besproken.

Deze masterproef kan opgedeeld worden in drie stukken. Het eerste deel is een API maken die de bestaande monitoringsinformatie beschikbaar maakt. Het is de bedoeling dat deze API een stevige basis vormt waarop andere applicaties kunnen gebouwd worden.

Het tweede deel is het maken van een monitoringsservice die de testbeds controleert. De data hiervan wordt dan via de API beschikbaar gemaakt. De monitoringsservice zal de huidige monitoringsservice vervangen.

Na uitwerking van eerste twee delen kan men eenvoudig kijken welke testbeds betrouwbaar zijn. Dit is echter niet voldoende. De testbeds hebben echter ook een educatief doel, namelijk het ondersteunen van labo's. De meeste docenten zijn echter niet altijd even overtuigd over het gebruik van testbed tijdens hun labo's. De voornaamste reden hiervoor is dat docenten geen garantie hebben dat het testbed de belasting, veroorzaakt door vele instructies van een grote groep, aankan.

Het laatste en derde deel zal dit probleem oplossen door een systeem te voorzien om loadtesten uit te voeren. Bij een loadtest wordt een testbed belast, waarna de reactie van het testbed geanalyseerd wordt. De resultaten van deze analyse bepalen vervolgens of het labo met X personen vlot kan verlopen.

Hoofdstuk 2

SFA-Architectuur

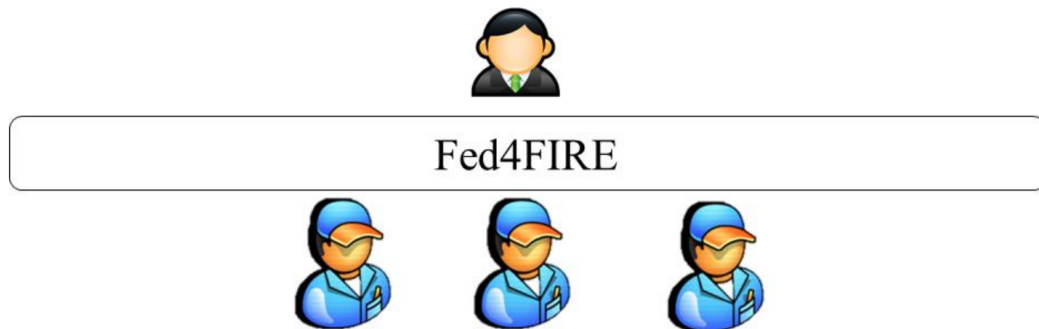
Deze masterproef is onderdeel van FIRE, een Europees onderzoeksproject naar een innovatief internet. Binnen FIRE maakt deze masterproef een monitoringsservice met een bijhorende API. FIRE gebruikt voor zijn testbedden de SFA architectuur. Deze architectuur is ontwikkeld om de configuratie en de aansturing van testbeds of aggregates over de hele wereld gelijk te maken. Hierdoor moeten onderzoekers maar één configuratie leren, waarmee ze vervolgens op elk testbed kunnen werken. Dit hoofdstuk gaat dieper in op SFA.

2.1 Doel

SFA (Slice-based Federation architecture) is een framework dat gebruikt wordt om testbeds aan te sturen (Peterson *et al.*, 2010). SFA is gebaseerd op FFA (First Federation Architecture) en wordt gebruikt om één van de FIRE doelstellingen, namelijk het delen van testbeds makkelijker maken, waar te maken. Doordat alle testbeds op een andere manier werken, is het voor een onderzoeker erg moeilijk om verschillende testbeds te gebruiken. Een onderzoeker moet eerst kennis maken met de specifieke configuratie van een testbed, alvorens hij ermee kan werken. Aangezien dit zeer tijdrovend is, is het noodzakelijk om de configuratie van testbeds overal gelijk te maken.

Een belangrijk begrip is een federation. Dit begrip heeft vooral te maken met autorisatie. Binnen een federatie worden testbeds, services en onderzoekers vertrouwd. De bedoeling van Fed4FIRE is dan ook de testbeds of aggregates binnen FIRE samen te voegen tot een federatie.

SFA is een standaard die door testbeds geïmplementeerd wordt. Hierdoor kan een onderzoeker die kennis heeft van SFA, direct ook werken met alle testbeds die er compatibel mee zijn. Zoals te zien is in Figuur 2.1 is de onderzoeker de klant en het testbed is de service provider.



Figuur 2.1: De onderzoeker (boven) is de klant, de testbeds (onderhouden door de onderste personen) is de service provider. Fed4FIRE voorziet de link tussen beide partijen.

SFA bezit een aantal functionaliteiten. Zo is voorzien dat het beleid van de eigenaar nageleefd wordt. SFA voorziet in mechanismen om dit te controleren.

SFA moet ook voorzien dat operators onderhoud kunnen uitvoeren. Hiervoor moet het mogelijk zijn om machines te verwijderen of te vervangen. Ook toevoegen van nieuwe machines moet mogelijk zijn.

Daarnaast moeten onderzoekers de mogelijkheid krijgen om experimenten aan te maken en de medewerkers voor het project te beheren. Hierbij hoort ook de autorisatie die gecontroleerd wordt door de eigenaars. Zo is het bijvoorbeeld mogelijk om maar een beperkt aantal mensen toegang te verlenen.

2.2 Entiteiten

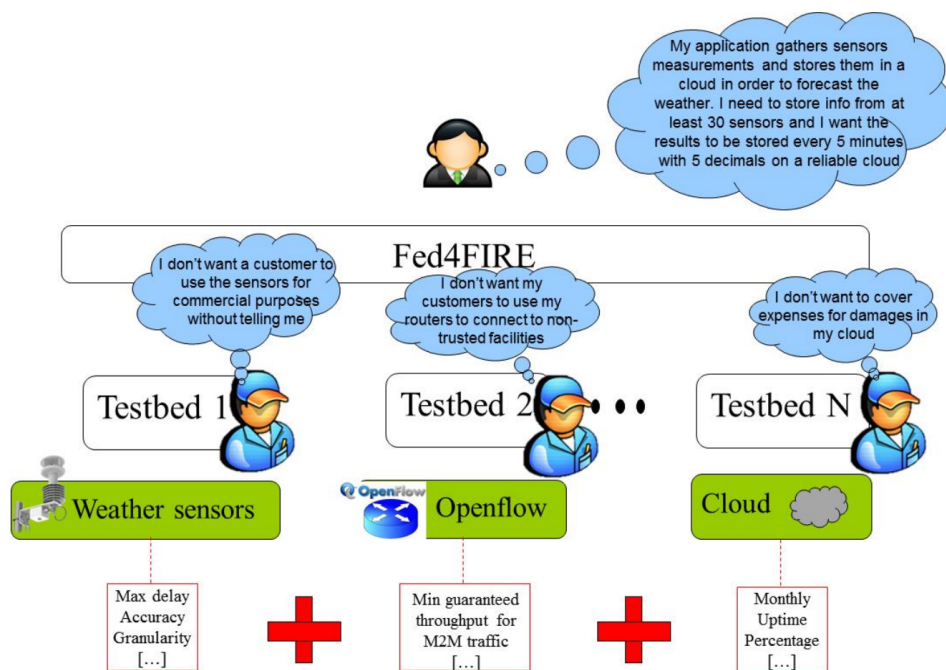
SFA herkent 4 entiteiten.

1. Owners
2. Operators
3. Researchers
4. Identity anchors / identity providers

Hierna volgt is een bespreking van elke entiteit met zijn verantwoordelijkheden.

2.2.1 Owners

De eigenaars of verantwoordelijken voor het testbed zijn verantwoordelijk voor de werking van zijn (deel van) het testbed. De Owners bepalen welke beleidsregels er van toepassing zijn. Deze beleidsregels worden aangeduid met SLA (Service Level Agreements). Figuur 2.2 geeft een voorbeeld van een aantal mogelijke beleidsregels. Zo kan het zijn dat de eigenaar van een testbed niet wil dat er commerciële testen gebeuren zonder dat hij daarvan op de hoogte is (Figuur 2.2 Testbed 1). Een ander voorbeeld is dat de eigenaar uitgaande verbindingen beperkt (Figuur 2.2 Testbed 2).



Figuur 2.2: Eigenaars bepalen het beleid van hun testbed.

2.2.2 Operators

Operators zorgen voor het onderhoud van het testbed. Dit onderhoud omvat o.a. herstelingswerken, beveiliging, voorkomen van schadelijke activiteiten.

2.2.3 Researchers

De onderzoeker is de klant. Hij gebruikt een testbed om er zijn experimenten op uit te voeren. Deze experimenten verlopen in het kader van zijn onderzoek.

2.2.4 Identity providers

Een identity provider of identity anchor is iemand die entiteiten rechten kan geven. Zo kan een identity provider een hoofdonderzoeker rechten geven om onderzoekers binnen zijn project te laten werken.

2.3 Opbouw van een testbed

Een testbed is opgebouwd uit meerdere onderdelen die hierna besproken worden.

2.3.1 Component

Een testbed bestaat uit vele onderdelen. Het primaire bouwblok van een testbed is een component. Een component is bijvoorbeeld een computer, router of switch. Indien de component een computer is, wordt deze ook een node genoemd. Een node is dus een computer, meestal binnen een testbed, verbonden met het netwerk.

2.3.2 Aggregate

Al deze componenten worden gegroepeerd in aggregates. Een aggregate is een verzameling componenten die onder eenzelfde beheerder valt. Zo is de virtual wall2 van iMinds een aggregate omdat het beheer van dit volledige testbed onder iMinds valt.

2.3.3 Aggregate manager

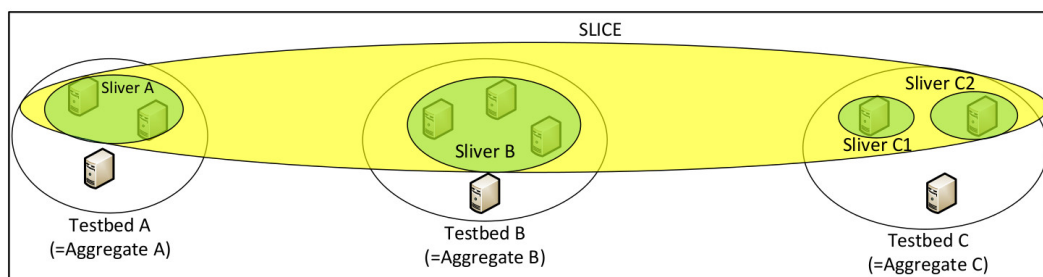
Elk aggregaat wordt beheerd door een AM (aggregate manager). De aggregate manager is een stuk software dat een interface aanbiedt aan onderzoekers. Via deze interface kan bijvoorbeeld 'plaats' gereserveerd worden om een experiment op te zetten. Een aggregate manager vervult taken zoals het toewijzen van van slices (zie later) aan onderzoekers of aan een experimenten.

2.3.4 Sliver

Een component kan echter ook gemultiplexed worden zodat er meerdere experimenten tegelijk op kunnen draaien. Dit kan door bijvoorbeeld virtualisatie toe te passen. Het 'stuk' van een component wordt een sliver genoemd.

2.3.5 Slice

Een slice is een verzameling van slivers. Een slice is een abstract begrip dat omschreven kan worden als een container waarin een experiment draait. Vanuit het perspectief van de onderzoeker komt dit overeen met de testopstelling die hij ter beschikking heeft. Vanuit het perspectief van de owner of eigenaar van het testbed is dit een administratieve opdeling om bij te houden welke testen er waar gebeuren. Figuur 2.3 geeft een voorbeeld van een slice. Hierbij is duidelijk dat een slice over meerdere aggregates kan lopen. Een slice kan over meerdere aggregates heen kan lopen. Deze aggregaten hebben elk een eigen beheerder en SLA (Service Level Agreements). Ook is op Figuur 2.3 duidelijk te zien dat een slice opgebouwd is uit slivers.

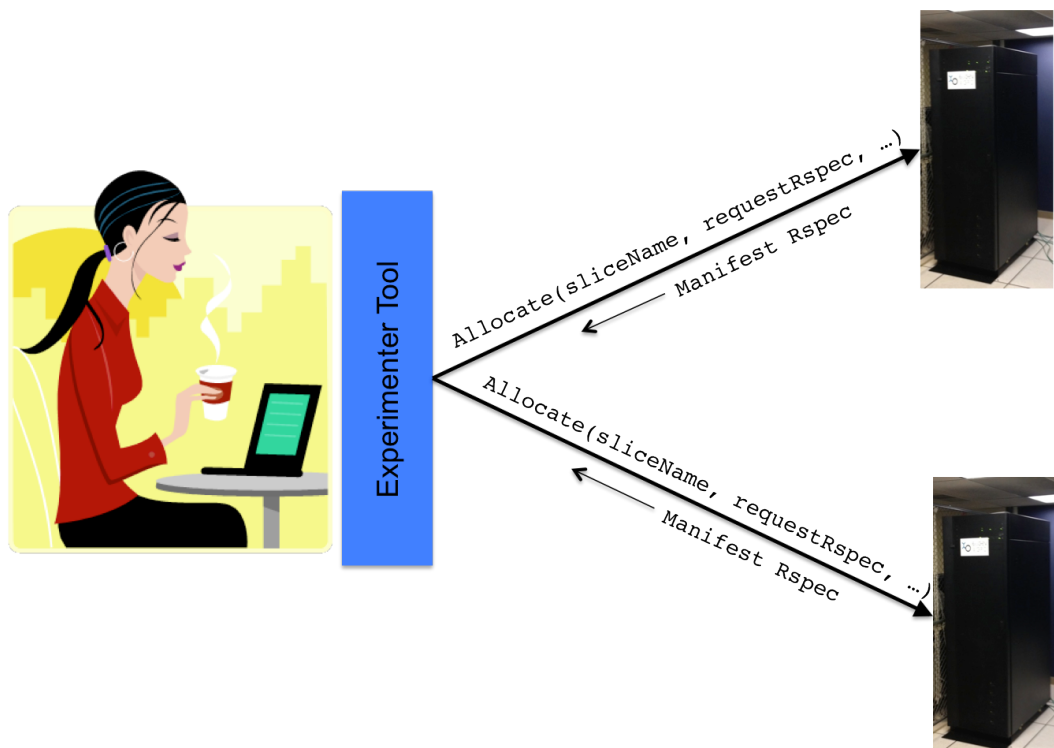


Figuur 2.3: Een slice (geel) bestaat uit een verzamling slivers (groen).

2.4 Communicatie met testbed via RSpec

Een RSpec (Resource Specification) , is een XML file die een proefopstelling beschrijft (GENI, 2014d). Het voordeel van dit formaat in plain tekst is dat onderzoekers zeer eenvoudig experimenten kunnen herhalen. Doordat een RSpec een volledige beschrijving is van een proefopstelling, is dit een meerwaarde in wetenschappelijke verslagen. RSpecs kunnen opgedeeld worden in drie soorten.

De eerste soort is een request RSpec. Een request RSpec beschrijft welke resources een onderzoeker wil gebruiken in zijn experiment. De AM (aggregate manager) antwoordt hierop met een manifest RSpec, zoals beschreven in Figuur 2.4 . Deze RSpec beschrijft de resources die gealloceerd zijn voor het experiment. Merk op dat dit proces transparant gebeurt, de meeste jFed tools zullen automatisch een RSpec genereren. Toch is het mogelijk om RSpecs zelf aan te maken, wat vooral voor complexere testopstellingen gebeurt.



Figuur 2.4: Een onderzoeker, of de tool die hij gebruikt stuurt eerst een request RSpec en krijgt vervolgens een manifest RSpec terug.

De derde soort RSpec, de advertisement RSpec, wordt gebruikt bij de listResourcetest. De listResourcetest komt later in de scriptie aan bod. Een advertisement RSpec lijst alle resources op die beschikbaar zijn op een testbed.

Hoofdstuk 3

Vooronderzoek

Zoals eerder besproken, maakt deze masterproef een monitoringsAPI in opdracht van het onderzoekscentrum iMinds. De monitoringsAPI heeft als doel de resultaten van de achterliggende monitoringsservice aan te bieden. De monitorService zal op zijn beurt alle aggregaten (testfaciliteiten) binnen FIRE. Dit hoofdstuk beschrijft de bestaande situatie en gaat daarbij in op de werking van de FIRE en GENI monitor. De FIRE monitor is de monitor die bij aanvang van de masterproef de monitoring verzorgde. De GENI monitor verzorgt de monitoring binnen het GENI project.

3.1 FIRE Monitor

iMinds heeft een monitoringsservice gemaakt die al enige tijd draait (Vermeulen, 2014), de FIRE monitor. Deze monitoringsservice is ruimer dan de monitoring van testbeds, ook het monitoren van experimenten wordt door deze service afgehandeld (Vermeulen, 2014). Hierna wordt de FIRE monitor besproken zoals ze was bij aanvang van de masterproef.

3.1.1 Componenten

De monitoringsservice bestaat uit verschillende componenten. De eerste component is de Facility monitoring. Deze monitoring wordt gebruikt bij de FLS (first level support). De first level support heeft als doel om de basistaken van monitoring af te handelen. De voornaamste test is de pingtest die kijkt of een testbed nog online is. De aggregate bepaalt zelf welke testen er uitgevoerd worden.

De tweede component is de infrastructure monitoring. Deze component is gericht op componenten binnen een experiment. De verzamelde gegevens omvatten o.a. het aantal verstuurde pakketten, het aantal verloren pakketten, de cpu-load,

De derde component is de OML measurement library. Deze bibliotheek laat het toe dat een onderzoeker zijn eigen monitoring framework gebruikt om de metingen van zijn experiment te doen.

Deze masterproef richt zich op de Facility monitoring. De tweede en de derde component zijn hier minder relevant en worden verder buiten beschouwing gelaten. De monitoringsservice waarnaar verwezen wordt, is bijgevolg de Facility Monitoring.

De monitoringsservice (Facility Monitoring) is opgedeeld in een aantal stukken. Het eerste stuk is de FLS-monitor (First Level Support). Dit heeft tot doel actuele informatie weer te geven over de status van het testbed en is beschikbaar op <https://flsmonitor.fed4fire.eu/>, zie Figuur 3.1 .

Fed4FIRE First Level Support Monitoring					
Testbed Name	Ping latency (ms)	GetVersion Status	Free Resources	Internal testbed monitoring status	Last check internal status
BonFIRE	31.17	N/A	N/A	ok	2014-03-25 11:16:09+01
Fuseco	16.83	ok	1	ok	2014-03-25 11:16:03+01
Koren	284.47	N/A	N/A	N/A	N/A
NETMODE	67.33	ok	20	ok	2014-03-25 11:13:22+01
NITOS Broker	73.03	ok	38	ok	2014-03-25 11:15:01+01
NITOS SFAWrap	31.29	ok	65	N/A	N/A
Norbit	N/A	N/A	N/A	ok	2014-03-25 11:10:29+01
Ofelia (Bristol island)	11.89	N/A	N/A	ok	2014-03-25 11:10:02+01
Ofelia (I2CAT island)	N/A	N/A	N/A	ok	2014-03-25 11:10:02+01
Planetlab Europe	32.11	ok	285	ok	2014-03-25 11:15:02+01
SmartSantander	53.82	ok	0	ok	2014-03-25 11:10:01+01
Virtual Wall	0.21	ok	23	ok	2014-03-25 11:14:39+01
w-lab.t 2	20.33	ok	68	ok	2014-03-25 11:14:58+01

Figuur 3.1: FLS testbed monitoring

Figuur 3.1 geeft een beeld van de FLS monitoringssite. De eerste kolom geeft de naam van het testbed weer. Daarnaast wordt het resultaat van de laatste ping test getoond. De volgende 2 kolommen bevatten het resultaat van respectievelijk de getVersiontest en de free resources test. GetVersion geeft aan of de AM (aggregate manager) nog werkt terwijl de kolom free resources aangeeft hoeveel resources er nog beschikbaar zijn. De vorm van deze testen is relatief eenvoudig, aangezien er slechts een enkelvoudig resultaat wordt teruggegeven. De laatste 2 kolommen zijn van minder belang.

Het tweede deel van de monitoringsservice, nightly login testing, bevat complexere testen. Deze testen worden typisch 1 tot 2 keer per dag uitgevoerd. Deze testen zijn diepgaander dan de FLS-monitor. Een belangrijke test die hier gebeurt is de logintest. Hierbij wordt getest of het aanmelden op een testbed mogelijk is. Een andere test die uitgevoerd wordt, is de stitchingtest. Deze kijkt of het mogelijk is om een netwerk op te zetten tussen de verschillende testbeds. Zie Figuur 3.2, waar de resultaten van stitching testen weergegeven worden. Deze resultaten zijn vereenvoudigd tot 2 tussenresultaten.

jFed Automated Scenario Tests								
Options: Show code tag								
Category	Test Name	Last Test Start Time (CET)	Last Test Duration	Last Partial Success	Last Full Success	Time since last Failure	Last Log	History
login	Fuseco	2014-03-25 09:17:47	14 seconds	FAILURE	FAILURE		log	history
login	InstaGeni BBN	2014-03-25 09:18:01	3 minutes and 6 seconds	SUCCESS	SUCCESS	9 days and 13 hours	log	history
login	InstaGeni Clemson	2014-03-25 09:21:08	45 seconds	WARN	WARN		log	history
login	InstaGeni GATech	2014-03-25 09:21:54	2 minutes and 24 seconds	SUCCESS	SUCCESS	27 days and 14 hours	log	history
login	InstaGeni Illinois	2014-03-25 09:24:20	3 minutes and 27 seconds	SUCCESS	SUCCESS	5 months and 11 days	log	history
login	InstaGeni Kettering	2014-03-25 09:27:50	10 seconds	FAILURE	FAILURE		log	history
login	InstaGeni MAX	2014-03-25 09:28:00	2 minutes and 55 seconds	SUCCESS	SUCCESS	2 months and 9 days	log	history

Figuur 3.2: Resultaten van de stitching test

Zoals blijkt uit Figuur 3.3, is het ook mogelijk om de geschiedenis van deze testen op te vragen. In Figuur 3.3 is de geschiedenis van de stitching test tussen wall1 en wall2 weergegeven.

jFed Automated Scenario Tests						
Options: Show code tag						
Category	Test Name	Test Start Time	Test Duration	Partial Success	Full Success	Log
stitching	Stitching vwall1 - vwall2	2014-04-21 03:50:33	6 seconds	FAILURE	FAILURE	log
stitching	Stitching vwall1 - vwall2	2014-04-21 04:07:34	2 minutes and 27 seconds	FAILURE	FAILURE	log
stitching	Stitching vwall1 - vwall2	2014-04-21 04:18:35	9 minutes and 46 seconds	SUCCESS	FAILURE	log
stitching	Stitching vwall1 - vwall2	2014-04-21 10:14:35	9 minutes and 26 seconds	SUCCESS	FAILURE	log
stitching	Stitching vwall1 - vwall2	2014-04-21 20:13:10	20 minutes	FAILURE	FAILURE	log
stitching	Stitching vwall1 - vwall2	2014-04-21 20:33:29	3 minutes and 13 seconds	SUCCESS	FAILURE	log
stitching	Stitching vwall1 - vwall2	2014-04-21 21:55:49	3 minutes and 19 seconds	SUCCESS	FAILURE	log
stitching	Stitching vwall1 - vwall2	2014-04-21 23:12:05	3 minutes and 17 seconds	SUCCESS	SUCCESS	log
stitching	Stitching vwall1 - vwall2	2014-04-22 10:13:13	3 minutes and 17 seconds	SUCCESS	SUCCESS	log
stitching	Stitching vwall1 - vwall2	2014-04-22 20:22:31	3 minutes and 14 seconds	SUCCESS	SUCCESS	log
stitching	Stitching vwall1 - vwall2	2014-04-23 10:20:53	3 minutes and 16 seconds	SUCCESS	SUCCESS	log

Figuur 3.3: Geschiedenis van stitching test tussen wall1 en wall2

3.1.2 Testen

De FLS monitor voert 3 soorten testen uit.

1. Een ping test die kijkt of een testbed nog online is. Dit is een eenvoudig ping commando dat regelmatig uitgevoerd wordt.
2. Een getVersion call naar de AM (aggregate manager) API.
Doordat de getVersion call geen authenticatie vereist, wordt deze test gebruikt om te testen of een AM nog werkt.
3. De listResources test.
Hiermee wordt gekeken hoeveel resources er nog beschikbaar zijn. Indien dit nul is, is het niet mogelijk om nieuwe testen te starten.

Daarnaast zijn er nog een aantal testen gedefinieerd. De belangrijkste zijn hier vermeld.

1. De login test:
deze test zal proberen om aan te melden op een testbed.
2. Stitching test:
Deze test is redelijk complex. Vereenvoudigd zal een stitching test 2 verschillende testbeds met elkaar verbinden. Hiervoor wordt er ingelogd op beide testbeds en vervolgens wordt er op elk testbed een node gereserveerd. Dan worden deze nodes met elkaar verbonden en probeert men tussen de nodes te pingen.

3.1.3 Werking FLS monitor

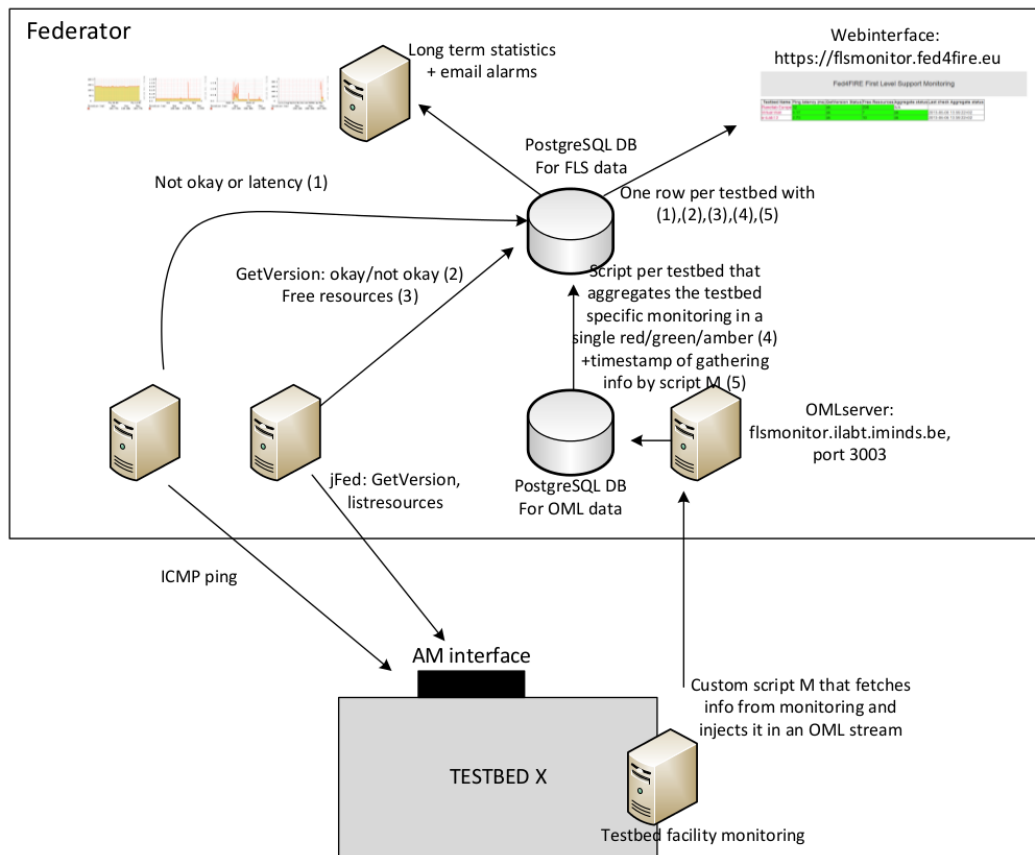
Figuur 3.4 geeft weer welke gegevens er opgevraagd worden. De eerste kolom geeft de naam van het testbed weer. De tweede kolom geeft de latency weer. De derde en de vierde kolom komen overeen met de getVersion call en de listResources call. De 5e kolom houdt de interne status van het testbed bij, dit wordt door het testbed zelf ingevuld en doorgegeven. Ten slotte is er nog een timestamp die aangeeft wanneer de laatste update verlopen is. Indien een timestamp te oud wordt, duidt dat op problemen met de monitor die draait op het testbed.

Fed4FIRE First Level Support Monitoring					
Testbed Name	Ping latency (ms)	GetVersion Status	Free Resources	Internal testbed monitoring status	Last check internal status
BonFIRE	31.17	N/A	N/A	ok	2014-03-25 11:16:09+01
Fuseco	16.83	ok	1	ok	2014-03-25 11:16:03+01
Koren	284.47	N/A	N/A	N/A	N/A
NETMODE	67.33	ok	20	ok	2014-03-25 11:13:22+01
NITOS Broker	73.03	ok	38	ok	2014-03-25 11:15:01+01
NITOS SFAWrap	31.29	ok	65	N/A	N/A
Norbit	N/A	N/A	N/A	ok	2014-03-25 11:10:29+01
Ofelia (Bristol island)	11.89	N/A	N/A	ok	2014-03-25 11:10:02+01
Ofelia (i2CAT island)	N/A	N/A	N/A	ok	2014-03-25 11:10:02+01
Planetlab Europe	32.11	ok	285	ok	2014-03-25 11:15:02+01
SmartSantander	53.82	ok	0	ok	2014-03-25 11:10:01+01
Virtual Wall	0.21	ok	23	ok	2014-03-25 11:14:39+01
w-Lab.t.2	20.33	ok	68	ok	2014-03-25 11:14:58+01

Figuur 3.4: FLS testbed monitoring

Figuur 3.5 op volgende pagina beschrijft de werking van de FLS monitor. Punt 1 stelt de pingtest voor tussen de monitor en het testbed. Hierbij wordt gekeken of de latency beneden een waarschuwwaarde valt. De ping test verloopt rechtstreeks naar het testbed en niet via de AM. Punt 2 en 3 zijn de getVersion en listResources call naar de AM op het testbed. Al deze gegevens worden bijgehouden in een PostgreSQL databank. Punt vier gaat over de interne status van het testbed. Dit wordt echter niet geïmplementeerd in de masterproef.

Rechtsboven is de webview weergegeven. Deze haalt de resultaten rechtstreeks uit de databank. Linksboven is de federator weergegeven. Deze wordt gebruikt door de OML measurement library, een andere monitoringscomponent binnen FIRE die niet relevant is voor deze masterproef.

**Figuur 3.5:** Werking van de FLS monitor

De FIRE monitor heeft een centrale database waarin de monitorinformatie zit. Deze database is de verbinding tussen de website en de resultaten van de test. Langs de ene kant wordt de uitvoer van de testen in de databank geüpdatet. Langs de andere kant worden deze gegevens opgevraagd door de website. Het is echter zo dat de FLS monitor zelf geen lange termijn statistieken bijhoudt. Dit komt doordat er bij het uitvoeren van testen geen nieuwe lijnen toegevoegd worden aan de databank. In plaats daarvan wordt de lijn geüpdatet.

3.2 Tekortkomingen de FIRE monitor bij aanvang

De facility monitoring component van de FIRE monitor werkt wel, maar vertoont toch enkele problemen. Deze worden hierna besproken.

3.2.1 Bereikbaarheid van de gegevens

Het voornaamste probleem is niet de structurering van de gegevens, maar de bereikbaarheid. De gegevens zijn enkel bereikbaar via de webinterface, of rechtstreeks via de databank. Dit maakt het moeilijk voor nieuwe ontwikkelingen om deze gegevens te gebruiken. Deze masterproef lost dit probleem op door het gebruik van een monitoringsAPI. Deze zorgt ervoor dat de gegevens vlot beschikbaar zijn via aanvragen aan de API.

Het ontwerpen van deze monitoringsAPI maakt deel uit van de masterproef. Deze API moet een vlotte toegang tot de resultaten garanderen. Ook moet het mogelijk zijn om deze resultaten te filteren. Zo is het bijvoorbeeld mogelijk om resultaten te filteren op datum of testbed. Het is de bedoeling dat alle communicatie met de achterliggende databank via de API verloopt. De websites die momenteel bestaan (zie Figuur 3.1), zullen nu niet meer rechtstreeks contact maken met de databank. In plaats daarvan zal de API gebruikt worden als datalaag.

Ook het toevoegen van nieuwe resultaten door de monitoringsservice verloopt via de API. Dit heeft als voordeel dat de API complexere zaken zoals foutafhandeling kan afhandelen. Een bijkomend voordeel is dat de databank niet extern bereikbaar moet zijn. Authenticatie kan in een hogere laag afgehandeld worden. Authenticatie zelf vormt geen onderdeel van de masterproef.

3.2.2 Structuur van de gegevens

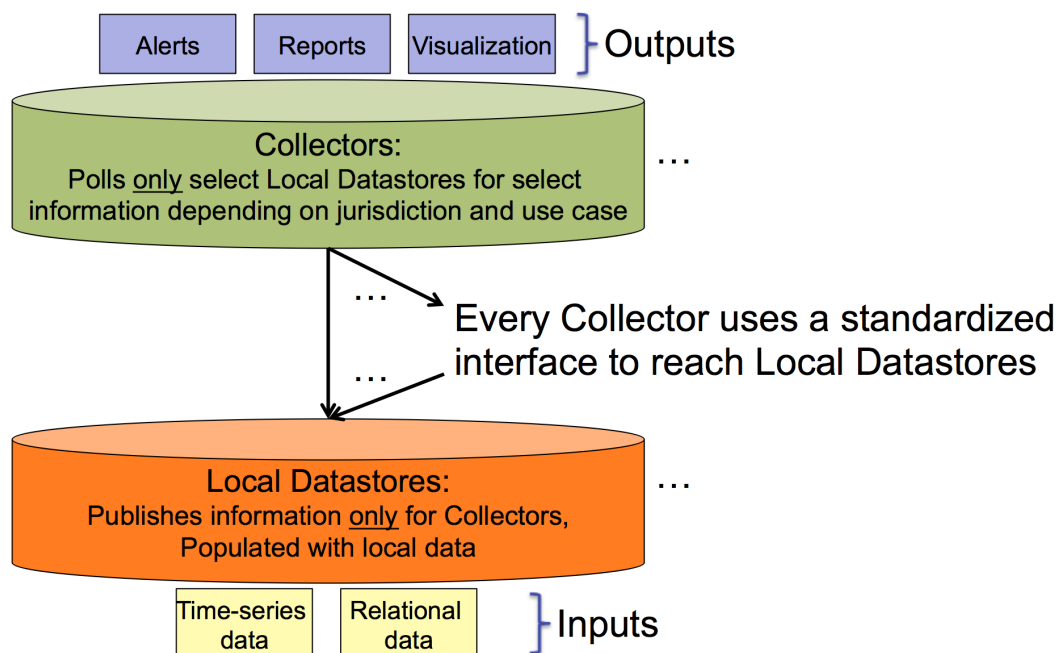
Naast de bereikbaarheid is er ook een structureel probleem. De testen van de FLS monitor hebben een eenvoudige structuur. De testen zijn eenvoudig omdat ze slechts een beperkt aantal parameters nodig hebben en een enkelvoudig resultaat teruggeven. Zo geeft een pingtest de ping waarde terug. Een listResources test geeft het aantal beschikbare resources terug. Beide waarden zijn gewoon getallen, die opgeslagen worden in een kolom van de databank.

De nightly login testen zijn echter complexer. Deze bestaan uit meerdere opeenvolgende stappen, die elk een eigen tussenresultaat hebben. Hierdoor volstaat de vorige databank die maar ovr een kolom voor resultaten beschikt, niet meer. De FIRE monitor lost dit probleem op door een tweede databank te gebruiken. Deze databank slaat niet alle tussenresultaten op, maar vereenvoudigd deze resultaten tot 2 tussenresultaten. Hierdoor is het moeilijk om snel een proper overzicht van alle tussenresultaten te krijgen. Voor deze informatie moet men zich wenden tot de log.

3.3 Integratie met GENI monitor

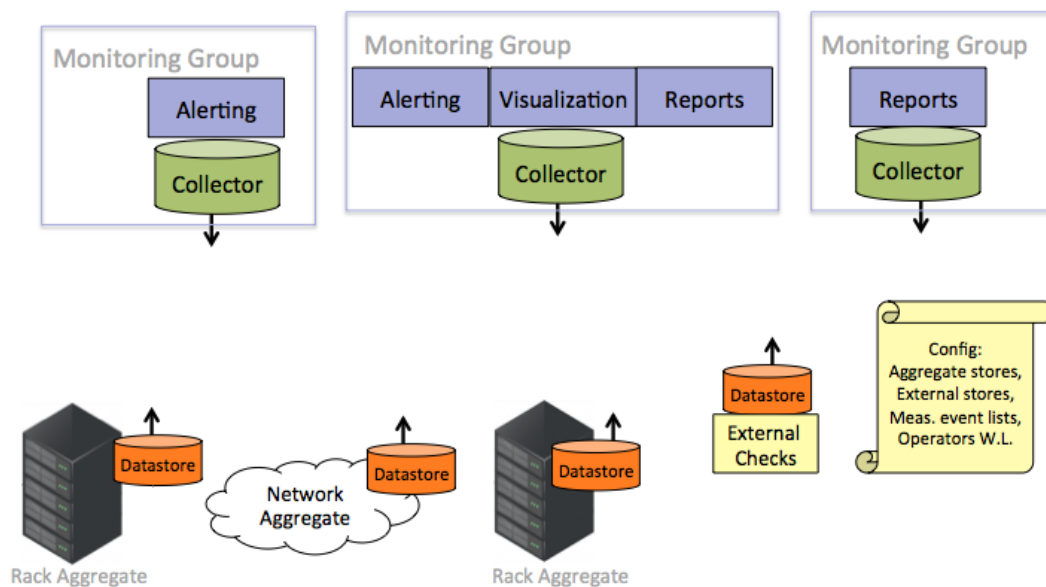
Als er binnen FIRE een nieuw monitoringAPI gemaakt moet worden, is het nuttig om onderzoek te doen naar bestaande oplossingen hiervoor. Zo is er binnen GENI ook een bestaand monitoringsframework (GENI, 2014b), dat hieronder uitgelegd wordt.

GENI (Global Environment for Network Innovations) is een Amerikaans project dat, net zoals Fed4FIRE, een groot aantal testbeds omvat. Dit lab bestaat uit meerdere onderzoekscentra. Aangezien GENI en FIRE beide bezig zijn met onderzoek naar innovatieve netwerk- en internetontwikkelingen, is de samenwerking tussen beide partijen vanzelfsprekend.



Figuur 3.6: Principe geni datastore

Testbeds binnen GENI maken ook gebruik van een Slice federation architectuur. GENI heeft ook een monitoring framework ontwikkeld. Dit framework werkt met meerdere databronnen of datastores. Dit principe is geschetst in Figuur 3.6. Elk testbed of groep van testbed heeft een datastore (oranje). Deze verzamelt monitoringsdata die ze via een API aanbieden. Deze API wordt vervolgens aangesproken door een collector (groen) die de gegevens verzamelt.



Figuur 3.7: Een collector kan verschillende delen data ophalen.

Een collector werkt voor een applicatie en pollt meerdere testbeds. De data die hierbij opgehaald wordt, is afhankelijk van wat de applicatie nodig heeft. Dit is beschreven in Figuur 3.7. De boxen bovenaan zijn de monitoringapplicaties. Deze hebben elk verschillende gegevens nodig, weergegeven in blauw. Elke monitoringsapplicatie heeft een collector (groen). De collector haalt de nodige gegevens op van een datastore (oranje).

Doordat GENI en FIRE gelijkaardig doelstellingen hebben is het de bedoeling dat beide projecten samenwerken. Hierdoor kunnen ontwikkelingen gedeeld worden. Daarnaast zijn er ook meer testbed beschikbaar als beide projecten samenwerken. Het is dan ook handig om de monitorAPI compatibel te maken met de GENI monitoringsservice. Daarvoor moet de monitoringAPI een datastore vormen. Concreet houdt dit in dat het antwoord van de resultaten aan een vastgelegde structuur moet voldoen. Omdat de huidige FIRE monitor veel meer informatie ter beschikking heeft dan de GENI datastore, zal deze masterproef kijken wat er geïntegreerd kan worden en welke uitbereidingen daarbij nodig zijn.

3.4 Besluit

De vorige service werkte wel, maar was niet voorzien op de komst van complexere testen. Er zijn twee grote problemen: enerzijds de bereikbaarheid van de gegevens, anderzijds de structuur. Het eerste probleem is opgelost door het maken van een API. Het tweede probleem is opgelost door het uitbouwen van een complexere databank. Deze databank houdt zowel de resultaten als de configuratie van de testen bij. Als uitbreiding is er de integratie met GENI. Hierbij zal onderzocht worden wat er mogelijk is en waarvoor er nog uitbreidingen nodig zijn.

Hoofdstuk 4

Ontwerp monitor

De masterproef maakt een monitoringssysteem bestaande uit een monitoringsservice en een monitoringsAPI die deze data beschikbaar stelt. De masterproef bestaat uit verschillende projecten. De kern van deze projecten is de API, de API biedt monitor informatie aan en vormt tevens de verbinding tussen alle andere projecten. Om aan monitorinformatie te komen is een monitoringsservice ontworpen. Deze service zal aggregates controleren en de resultaten opslaan in de databank. Tenslotte is een website ontworpen om de informatie weer te geven.

4.1 Structuur

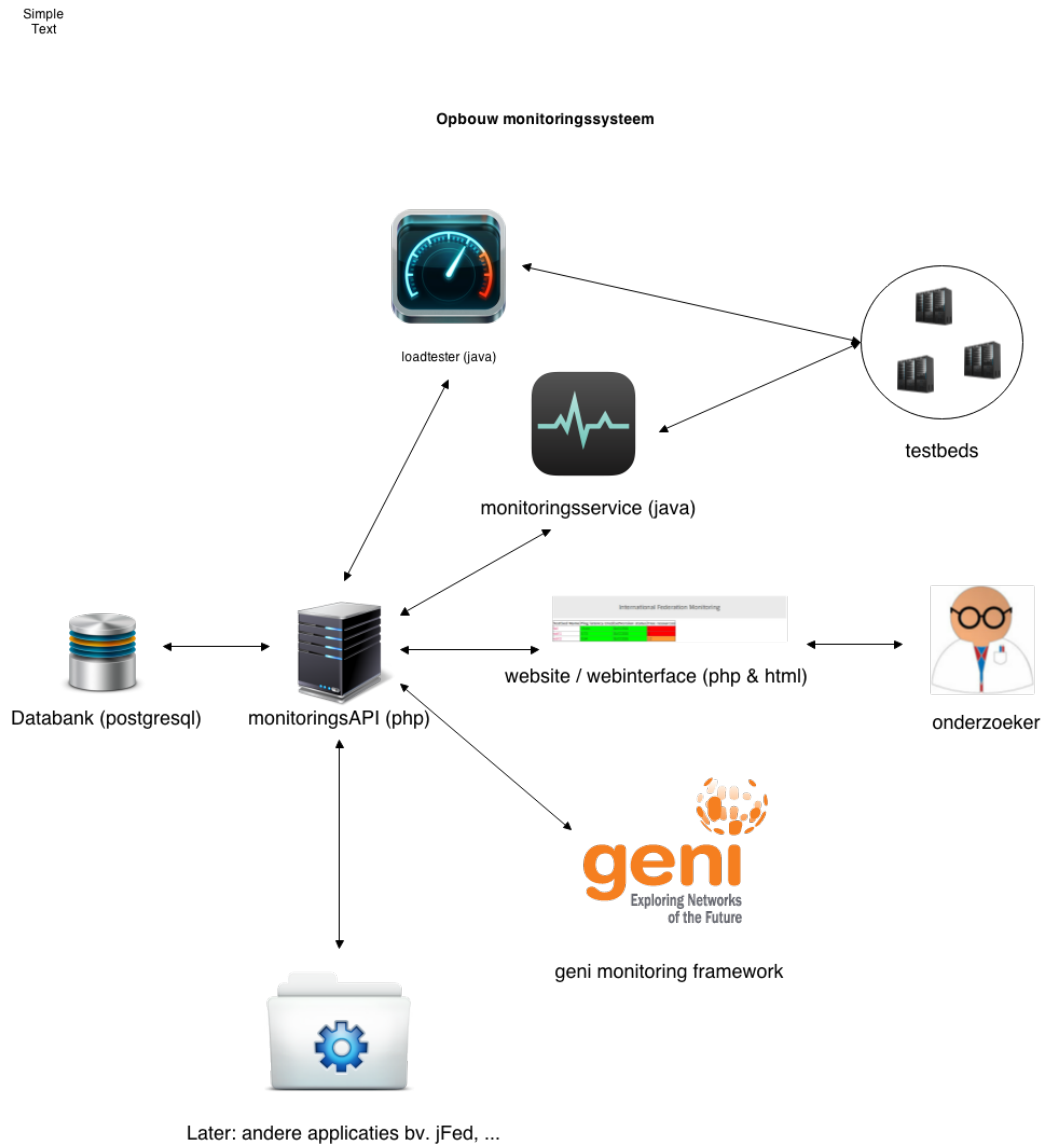
De masterproef zal een monitor maken bestaande uit een aantal projecten. Volgende sectie bespreekt hun verband en waarvoor elk deel verantwoordelijk is.

1. Een databank die alle data bijhoudt.
2. Een monitoringsAPI die de kern vormt. Alle andere projecten zijn verbonden via de API.
3. Een monitoringsservice die de testen uitvoert.
4. Een loadtester om testbeds te belasten.
5. Een website om de monitoringsinfo weer te geven.

Daarnaast zijn er nog 2 projecten weergegeven. Deze projecten zijn niet door deze masterproef ontwikkeld. Ze zijn vermeldt vanwege hun verband met de masterproef.

1. Geni monitoringframework, hiervoor wordt de interface van een datastore geïmplementeerd.
2. Later: andere applicaties bv.jFed, ... Dit duidt erop dat de monitoringsAPI een basis is waarop andere applicaties kunnen verder bouwen. Zo is het mogelijk dat de monitoringsinformatie in de toekomst geïntegreerd wordt in de primaire gebruikers interface van jFed.

Figuur 4.1 geeft een schematische weergave van de verschillende delen van de masterproef die hierboven uitgelegd staan.



Figuur 4.1: De samenhang van de verschillende projecten in de masterproef.

De volgende pagina's geven een korte omschrijving van wat elk deel moet kunnen. De werking en concrete implementatie komen in een later hoofdstuk aan bod.

4.2 De API

Dit onderdeel vormt de kern die alle andere projecten aan elkaar bindt. De monitoringsAPI staat in voor de communicatie tussen de buitenwereld en de databank. Enerzijds worden er resultaten toegevoegd aan de databank. Deze resultaten zijn afkomstig van de monitorings-service. Anderzijds worden er resultaten opgevraagd uit de databank door zowel de website als de GENI collector. De monitoringAPI is verantwoordelijk voor het beheer van de databank. Alle communicatie met de databank zal via de API verlopen. Dit heeft als voordeel dat zaken zoals foutafhandeling maar een keer geïmplementeerd moeten worden. Voor de implementatie wordt PHP gebruikt in combinatie met een apache HTTP server.

4.3 De databank

De databank is verantwoordelijk voor het bijhouden van de informatie. Deze informatie kan opgedeeld worden als volgt:

1. Configuratie van testen:
 - (a) De testbeds die door de monitoringsservice gecontroleerd moeten worden.
 - (b) De logins die gebruikt worden voor authenticatie op de testbeds.
 - (c) De beschikbare testen: hierbij moet het mogelijk zijn om nieuwe testen toe te voegen zonder te veel verandering aan te brengen in de code.
 - (d) De planning van de testen, hierbij moet het mogelijk zijn om elke test met een verschillend interval uit te voeren.
2. Resultaten van testen: naast het bijhouden van de configuratie moeten ook resultaten van elke test bijgehouden worden.

De databank zit verborgen achter de monitoringsAPI. Alle communicatie met de databank verloopt via de monitoringsAPI. De databank is een PostgreSQL databank.

4.4 De monitoringsservice

Dit deel voert de testen uit. Eerst zullen de testen die uitgevoerd moeten worden opgevraagd worden aan de API. Vervolgens worden deze testen simultaan uitgevoerd. Hierbij wordt gebruikt gemaakt van een threadpool. De monitoringsservice wordt in Java gemaakt.

4.5 De loadtester

Een loadtest wordt gebruikt om te kijken welke belasting een testbed kan afhandelen. Een voorbeeld van een situatie waar dit nodig is, is wanneer een docent met een groep studenten het testbed wil gebruiken voor een labo. Hierbij zouden 50 studenten elk 2 computers gebruiken om tcp-congestie te testen. TCP-congestie wordt gebruikt om fileproblemen die ontstaan door het overlopen van buffers te verhelpen. Deze problemen ontstaan nadat een host meer informatie doorstuurt dan een tussenliggende router kan afhandelen. Hierdoor zal de buffer van de router overlopen, wat leidt tot verlies van pakketten. Aangezien het TCP protocol garandeert dat een overdracht zeker en compleet is, worden deze pakketten opnieuw verzonden. Als men echter de frequentie waarmee de pakketten verstuurd worden niet verlaagt, zal dit alleen maar leiden tot meer verloren pakketten.

Het probleem is dat de docent geen garantie heeft dat het testbed dit aankan. Hierdoor zijn vele docenten weerhoudend om testbeds te gebruiken voor labo's. Om aan te tonen dat het testbed wel een belasting van 50 studenten met elk 2 nodes aankan, wordt een stresstest uitgevoerd. De bedoeling van deze stresstest is om 50 logintesten tegelijkertijd uit te voeren. Deze zullen een belasting veroorzaken op het testbed die opgemeten en geanalyseerd wordt. Op basis van de resultaten kan de docent met een gerust hart gebruik maken van het testbed of net aangeraden worden om de groep op te delen. De loadtester zal een applicatie voorzien om het testbed te belasten.

4.6 De website

De layout van de vorige webservice is overgenomen, maar een aantal punten zijn aangepast. De nieuwe website geeft wel alle tussenresultaten weer in het overzicht. Voorts is ook de backend van de site vervangen door een aantal API-calls.

De bedoeling van deze website is een eenvoudig, maar duidelijk overzicht bieden. Hierbij moet een onderzoeker zeer snel de status van het testbed waarop hij werkt, kunnen raadplegen. Voor dit deel wordt gebruik gemaakt van javascript, html en PHP. Daarnaast wordt Apache gebruikt als HTTP server.

4.7 Het GENI monitoringframework

Het GENI monitoringsframework bestaat uit 2 delen. Het eerste deel is een datastore; dit is een locatie waar monitoringsinformatie beschikbaar is. Het tweede deel is een collector. Een

collector wordt gebruikt door een applicatie om de monitoringsinformatie die hij nodig heeft op te halen van de datastores.

Binnen het GENI project zijn er al mensen bezig met de beveiliging en weergaven van de monitoringsdata. Door de API compatibel te maken met de GENI monitor, kunnen deze ontwikkelingen in de toekomst eenvoudig overgenomen worden. Als laatste deel van de masterproef zal er gekeken worden welke integratie mogelijk is, en of er uitbreidingen nodig zijn aan de API's.

4.8 Toekomstige ontwikkelingen

Dit stuk geeft aan dat de monitoringsAPI verder gaat dan huidige toepassingen. Het is de bedoeling dat de monitoringsAPI de monitoringsinformatie toegankelijk maakt voor toekomstige ontwikkelingen. Een voorbeeld hiervan is de integratie van de monitoringsinformatie in jFed. Op deze manier zou een onderzoeker die met jFed werkt meteen kunnen zien welke testbeds betrouwbaar zijn. Vervolgens kan hij een betrouwbaar testbed gebruiken voor zijn testen op uit te voeren.

Hoofdstuk 5

Implementatie van de monitor

Deze masterproef maakt een monitoringsservice met een bijhorende monitoringsAPI die de monitoringsinformatie beschikbaar stelt. De API houdt de resultaten bij in een databank. Deze databank bevat de configuratiegegevens, de resultaten en de beschrijvingen van de verschillende testen. De API doorloopt voor elke aanvraag een opeenvolging van stappen. Zo wordt eerst de aanvraag geparsed, vervolgens wordt een query gemaakt en uitgevoerd. Het resultaat van deze query wordt omgevormd tot objecten die vervolgens geëncodeerd worden. De website zal deze geëncodeerde objecten eerste decoderen en vervolgens visualiseren. Het laatste project is de monitoringsservice. Deze service zal via de API de testen binnenhalen. Deze testen worden vervolgens uitgevoerd en het resultaat wordt teruggestuurd naar de API. Tenslotte is er ook de loadtest die gebruikt wordt om te kijken welke belasting een testbed aankan.

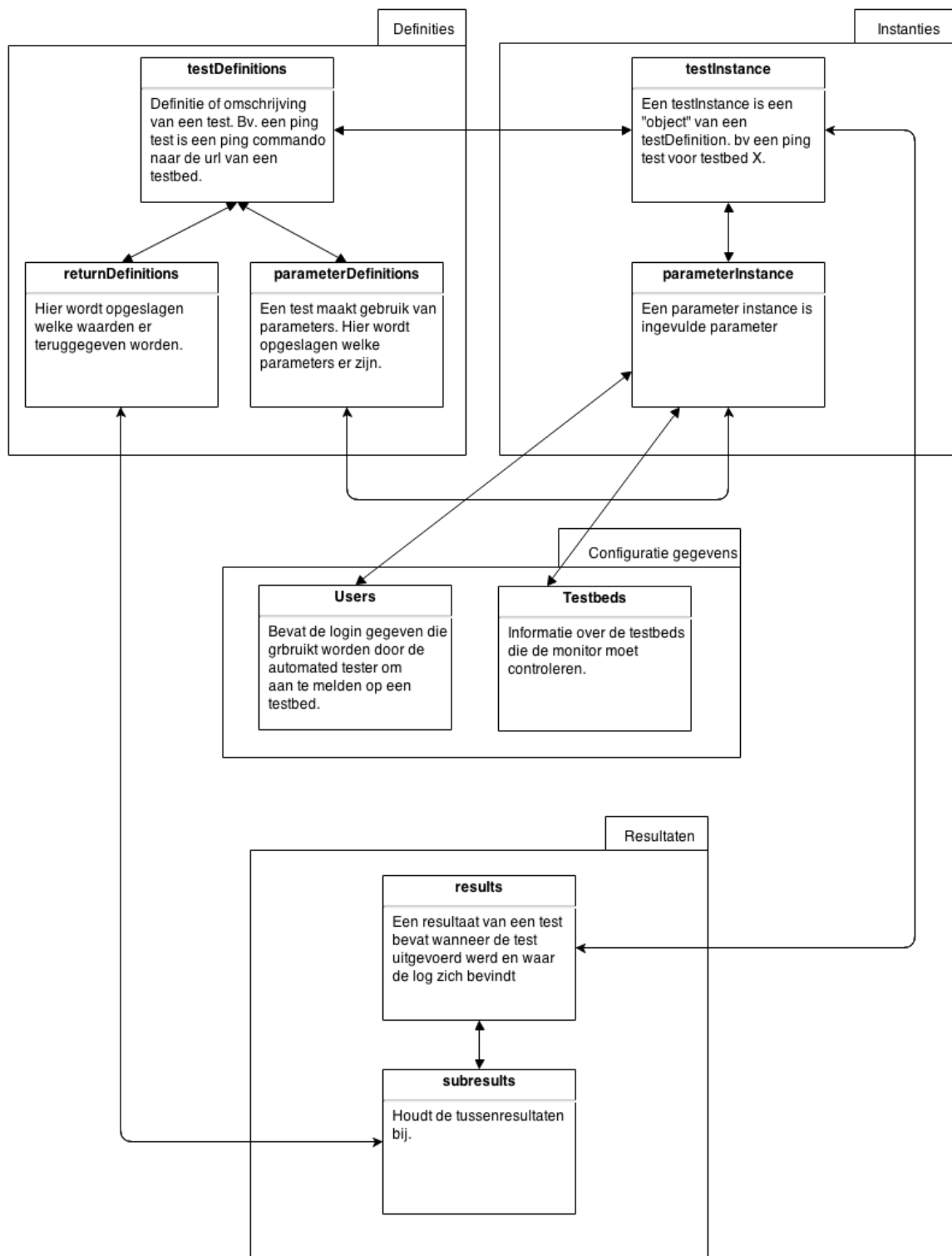
5.1 Databank

De databank bestaat uit meerdere tabellen die met elkaar verbonden zijn:

1. users: deze tabel houdt info bij over de login gegevens die gebruikt worden.
2. testbeds: deze tabel houdt info bij over de testbeds die gemonitord worden.
3. testDefinitions: deze tabel bevat beschrijvingen van de verschillende testen.
4. parameterDefinitions: deze tabel bevat per rij een beschrijving van een parameter.
5. returnDefinitions: deze tabel bevat een beschrijving van de waarden die teruggegeven worden.
6. testInstance: deze tabel bevat een object van een testDefinitie.
7. parameterInstance: de waardes van de parameters.
8. results: de resultaten
9. subResults: de tussenresultaten.

Dit alles is geschetst in Figuur 5.1 op volgende pagina. Hier zijn de tabellen gegroepeerd op basis van functionaliteit om een beter overzicht te behouden.

Structuur databank



Figuur 5.1: De structuur van de databank

5.1.1 Definities

De eerste groep tabellen bevat de definities. De definities bevatten een omschrijving van een test. Hierbij worden de parameters en de tussenresultaten ook opgeslagen. Deze worden opgenomen in een extra tabel om meer flexibiliteit toe te laten. Door de parameters en tussenresultaten in een andere tabel onder te brengen, is het mogelijk om verschillende testen met een variabel aantal tussenresultaten en parameters op te slaan.

5.1.2 Instanties

Deze tabel houdt de testinstanties bij. Een testinstantie is de test zelf. Als de vergelijking met objectgeoriënteerd programmeren gemaakt wordt, dan is een testDefinitie een klasse zelf en de instance is dan een object. Deze opsplitsing heeft het voordeel dat de beschrijving van een test apart opgeslagen kan worden. Het is vervolgens zeer eenvoudig om meerdere instanties aan te maken. Ook laat het systeem de nodige flexibiliteit toe om nieuwe definities aan te maken. Net zoals bij de definities zijn de ingevulde waarden hier ook ondergebracht in een aparte tabel. Dit is om dezelfde reden, namelijk het toelaten van een variabel aantal parameters.

5.1.3 Configuratiegegevens

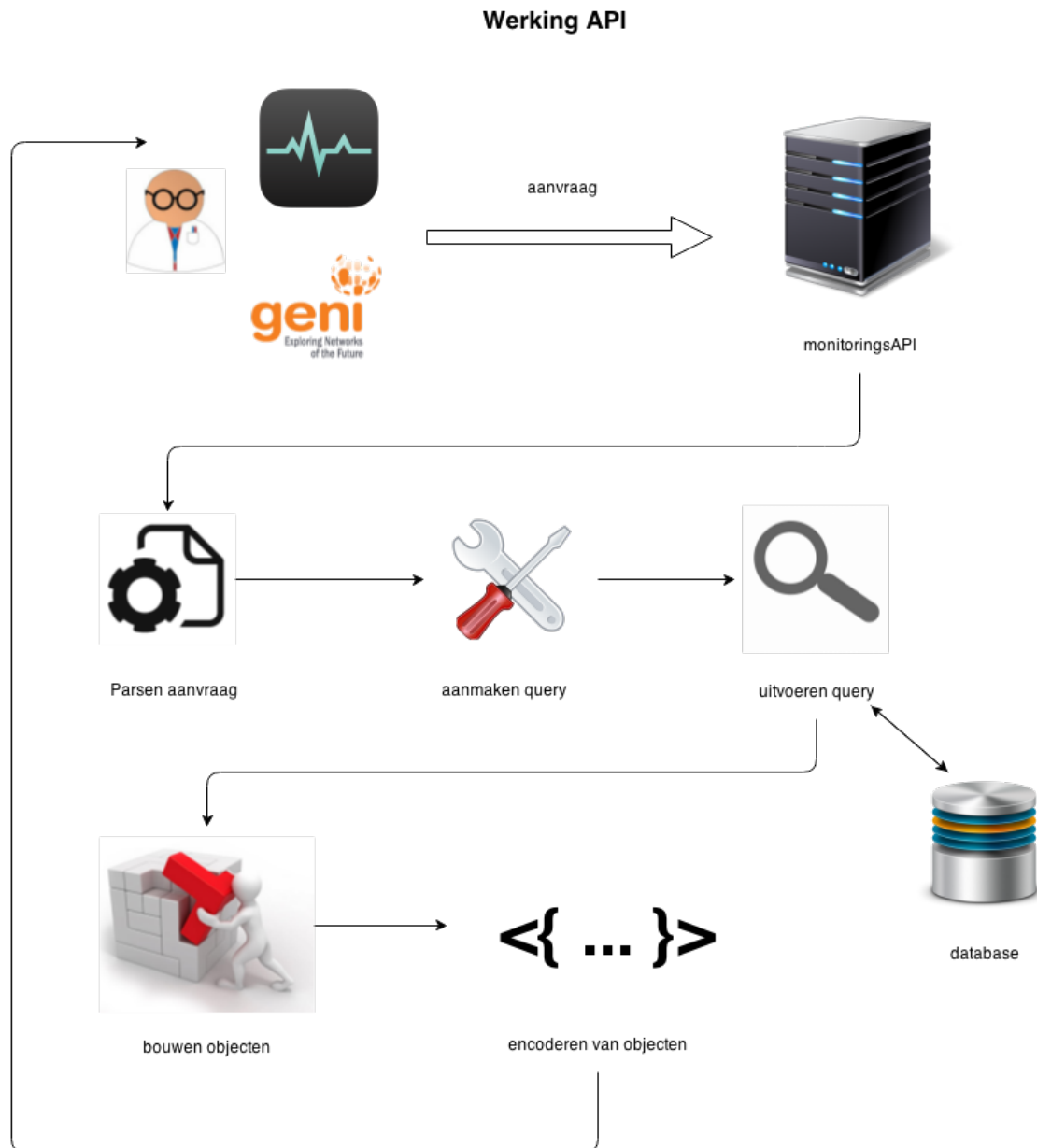
De configuratiegegevens bestaan uit de gebruikers en de testbeds. De gebruikers zijn belangrijk omdat ze de logininformatie bevatten die gebruikt worden door de automated tester om testen uit te voeren. Zo heeft een logintest een gebruiker nodig die de juiste rechten heeft op dat testbed. Naast de gebruikers is er ook de informatie over de testbeds. De testbeds hebben o.a. elke een eigen url, urn en naam. Deze informatie wordt in deze tabel ondergebracht. Zowel een gebruiker als een testbed kan/kunnen vervolgens opgegeven worden als een parameter van een testinstance.

5.1.4 Resultaten

Deze tabel houdt alle resultaten bij. Elk resultaat heeft een bijhorende logfile; deze wordt momenteel niet opgeslagen in de databank, maar wel apart als file georganiseerd in mappen op harde schijf. Het pad naar de logfile wordt vervolgens opgenomen in de databank. Dit heeft als voordeel dat de databank niet overvol geraakt met logfiles. Op die manier kunnen bijvoorbeeld alle tussenresultaten 6 maanden bijgehouden worden terwijl de logfiles maar voor 2 maanden bijgehouden worden.

5.2 Webservice / API

De webservice haalt informatie uit de databank op om ze om te vormen naar objecten. Dit verloopt in een aantal fasen, zoals weergegeven in Figuur 5.2.



Figuur 5.2: De werking van de API

5.2.1 Fasen

De stappen uit Figuur 5.2 die doorlopen worden zijn:

1. parsen van de aanvraag
2. opbouwen van de query
3. uitvoeren van de query
4. samenstellen van de objecten
5. encoderen van de objecten

De tekst hieronder zal kort een overzicht geven van elke stap.

5.2.2 Parsen aanvraag

Er zijn drie soorten aanvragen. De eerste behandelt het opvragen van configuratiegegevens. De tweede en derde soort zorgen voor het ophalen en toevoegen van resultaten. De laatste groep aanvragen is noodzakelijk voor de monitoringsservice en betreft het ophalen van testen en aanpassen van de planning. Een lijst van alle aanvragen met bijhorende parameters is terug te vinden in bijlage A.

Aanvragen van de eerste groep zijn:

1. testbed: geeft informatie weer over een of meerdere testbeds.
2. user: geeft informatie weer over een of meerdere users.
3. testDefinition: geeft informatie weer over de opbouw van een test.
4. testInstance: geeft informatie weer over de testen.

De tweede groep gaat over het opvragen van resultaten:

1. last: geeft de laatste resultaten weer.
2. list: geeft een lijst met resultaten weer.
3. q: wordt gebruikt voor het afhandelen van GENI aanvragen. Hierbij wordt de aanvraag geëncodeerd in JSON, het antwoord wordt geëncodeerd volgens GENI dataschema's (zie ook de referentie in bijlage A).

Een derde groep aanvragen verzorgt het uitvoeren van de testen en de planning van testen. Bij de planning is vooral het veld `nextrun` van belang. Dit veld geeft aan wanneer de test een volgende keer uitgevoerd moet worden.

1. `testInstance` gefilterd op `nextrun`: geeft de testen weer die uitgevoerd moeten worden op een zeker tijdstip.
2. `updateNextrun`: zal het moment waarop de volgende test uitgevoerd moet worden aanpassen.
3. `addResult`: zal een nieuw resultaat toevoegen aan de databank hiervoor met het type wel een HTTP-post request zijn.

5.2.3 Aanmaken query

In deze stap wordt de query aangemaakt; opgegeven filters worden vertaald naar SQL syntax. Bijvoorbeeld:

```
/last?testdefinitionname=ping,stitching
```

zal vertaald worden naar een `where` clause :

```
... where testdefinitionname IN (ping,stitching) ...
```

Dit wordt gedaan door de `QueryBuilder`, een interface die verschillende implementaties heeft om verschillende types van aanvragen te behandelen. Zo wordt een aanvraag afkomstig van GENI door een andere `QueryBuilder` afgehandeld dan een aanvraag van binnen FIRE.

5.2.4 Uitvoeren query

In deze stap wordt de query uitgevoerd met behulp van de `php-pgsql` module die de verbinding tussen PHPcode en een `postgreSQL` databank afhandelt.

5.2.5 Samenstellen van objecten

In deze stap wordt het resultaat van de query lijn per lijn overlopen en worden objecten gevormd. Om compabiliteit te verhogen worden er geen klassen zoals testbed, user en test aangemaakt, maar wordt alles direct opgeslagen in geneste array¹. Een geneste array is een array die arrays bevat. Deze arrays kunnen op hun beurt weer bestaan uit arrays en zo verder.

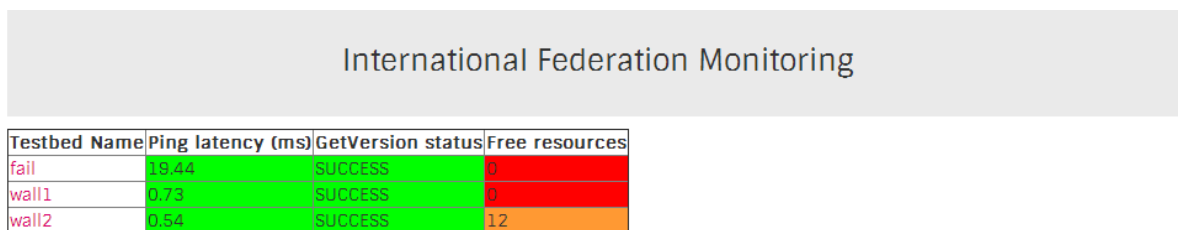
Deze manier van werken spaart tijd. De objecten worden direct in de 'vorm' van het dataschema gegoten. Een dataschema is een definitie van de vorm van het antwoord; concreter: hoe het antwoord er zal uitzien. Dit wordt gedaan door de fetcher. Opnieuw wordt hier gebruik gemaakt van overerving om andere dataschema's te ondersteunen.

5.2.6 Encoderen van objecten

De encoding is de laatste stap die het mogelijk maakt om objecten te versturen over een netwerk. De encoding wordt gedaan in JSON door de formatter. Door overerving kan hier gemakkelijk gezorgd worden voor extra coderingen bijvoorbeeld xml-encoding.

5.3 Website

De website bestaat uit een aantal verschillende interfaces waarvan de eerste de FLS (First Level Support) monitor is. Deze heeft als doel om voor elk testbed een snelle, eenvoudige statusweergave te voorzien. Een screenshot is te zien in Figuur 5.3.



The screenshot shows a web interface with a header 'International Federation Monitoring' and a table below it. The table has four columns: 'Testbed Name', 'Ping latency (ms)', 'GetVersion status', and 'Free resources'. The first row shows 'fail' with a latency of 19.44, status SUCCESS, and 0 free resources. The second row shows 'wall1' with a latency of 0.73, status SUCCESS, and 0 free resources. The third row shows 'wall2' with a latency of 0.54, status SUCCESS, and 12 free resources.

Testbed Name	Ping latency (ms)	GetVersion status	Free resources
fail	19.44	SUCCESS	0
wall1	0.73	SUCCESS	0
wall2	0.54	SUCCESS	12

Figuur 5.3: FLS monitor

¹Merk op dat een PHP array zowel een map als een array is.

Een volgende interface wordt gebruikt om detailresultaten weer te geven voor één testtype. In Figuur 5.4 wordt een overzicht gegeven van de login testen². Op deze figuur kan eenvoudig afgelezen worden dat de test voor het eerste testbed mislukt is en dat de oorzaak daarvan bij stap twee en vier (in rood) ligt. Diezelfde test is wel gelukt voor wall1 en wall2, maar met een waarschuwing bij stap 5.

International login2 Monitor										
TestName	Last execution (CET)	Last failure	Status							
fail	03/06/2014 - 13:45:16	00:12	0	1	2	3	4	5	6	7
wall1	03/06/2014 - 13:47:22	02:12	0	1	2	3	4	5	6	7
wall2	03/06/2014 - 13:47:03	01:51	0	1	2	3	4	5	6	7

Figuur 5.4: Overzicht van resultaten van de login testen.

Tenslotte is er nog een overzicht van de geschiedenis van een specifieke test. Dit is zichtbaar in Figuur 5.5. Wanneer de cursor over een tussenresultaat gaat, verschijnt er een tooltip³ waarop de naam van de subtest samen met de status wordt weergegeven⁴.

History of wall2										
Time execution (CET)	duration	Status								
03/06/2014 - 13:47:03	01:51	0	1	2	3	4	5	6	7	8
03/06/2014 - 01:42:03	01:54	0	1	2	3	4	5	6	7	8
02/06/2014 - 13:37:16	01:54	0	1	2	3	4	5	6	7	8
02/06/2014 - 01:32:05	01:54	0	1	2	3	4	5	6	7	8
01/06/2014 - 13:26:59	01:50	0	1	2	3	4	5	6	7	8
01/06/2014 - 01:22:01	01:49	0	1	2	3	4	5	6	7	8
31/05/2014 - 13:16:59	01:51	0	1	2	3	4	5	6	7	8
31/05/2014 - 01:11:59	01:50	0	1	2	3	4	5	6	7	8
30/05/2014 - 13:08:17	03:03	0	1	2	3	4	5	6	7	8
30/05/2014 - 01:02:35	02:25	0	1	2	3	4	5	6	7	8

Figuur 5.5: Geschiedenis van een login test.

Bij elke test is het mogelijk om de console-uitvoer te bekijken, deze is beschikbaar in de log. Naast de console-uitvoer genereert de automated tester ook een XML en HTML file met informatie over het verloop van de test. Deze zijn ook beschikbaar, zie Figuur 5.6a en Figuur 5.6b.

²Login2 in de titel is te verklaren omdat dit logintesten zijn die met de aggregate manager versie 2 werken.

³Maakt gebruik van powertip, beschikbaar op <http://stevenbenner.github.io/jquery-powertip/>

⁴Dit is ook mogelijk bij de weergave van een test.

```

<geni_code_id success>03731/geni_code_id success>
<start_time>2014-06-03T13:45:14.465+02:00</start_time>
<stop_time>2014-06-03T13:45:14.914+02:00</stop_time>
</apiinfo>
</apiinfo>
<description>
</description>
▼ <loglines>
  ▼ <logline>
    <text>
      The received RSpec has a length of 177852 characters, and lists 31 nodes.
    </text>
    <timeInfo>1401798915161</timeInfo>
    <type>NOTE</type>
    </logline>
  </loglines>
  ▼ <logline>
    <text>
      Found nodes for the following component_manager_id's: {unpublished:ID0=wall2.labst.unimib.it}
    </text>
    <timeInfo>1401798917177</timeInfo>
    <type>NOTE</type>
    </logline>
  </loglines>
  ▼ <logline>
    <text>
      Found potential fixed node: unpublishID=ID0=wall2.labst.unimib.it;name=benzene+n096-10b
    </text>
    <timeInfo>1401798917177</timeInfo>
    <type>NOTE</type>
    </logline>
  </loglines>
  ▼ <logline>
    <text>Start silver type found: emulab-openvnc</text>
    <timeInfo>1401798917177</timeInfo>
    <type>NOTE</type>
    </logline>
  </loglines>
</methodData>
<testName>test:ID0=unpublished:ID0=wall2.labst.unimib.it
<startTimeInfo>1401798912458</startTimeInfo>
<testSuccess>true</testSuccess>
<stopTimeInfo>1401798917177</stopTimeInfo>
<durationInfo>4719</durationInfo>
</method>

```

(a) De originele overview in XML formaat

Test Settings

```
Test User URN urn:publicid:IDN+wall3.ilabt.iminds.be:user#fctest
Test User Authority urn:publicid:IDN+wall3.ilabt.iminds.be:authority#cm
Tested Aggregate Manager (if applicable) urn:publicid:IDN+wall3.ilabt.iminds.be:authority#cm
Test Class be.iminds.ilabt.jfcd.localhost.apis.test.TestAggregateManager
Tested Method Only group "nodegroup" = "dependence"
Test Description
Many Aggregate Manager (Geni AM API v2) Tests. 2 slices and a silver will be created du
```

Overview

Total duration 111.214s from Tue Jun 03 13:45:11 CEST 2014 to Tue Jun 03 13:47:02 CEST 2014

- ✓ setUp
- ✓ testGetVersionXmlRpcCorrectness
- ✓ testListResourcesAvailableNoSlice
- ✓ testCreateSliceSliver
- ✓ testCreateSliver
- ⚠ testCreatedSliverBecomesReady
- ✓ checkManifestOnceSliverIsReady
- ✓ testNodeLogin
- ✓ testDeleteSliver

Details

```
setUp
SUCCESS
Test Class setup
```

(b) Overview in html formaat

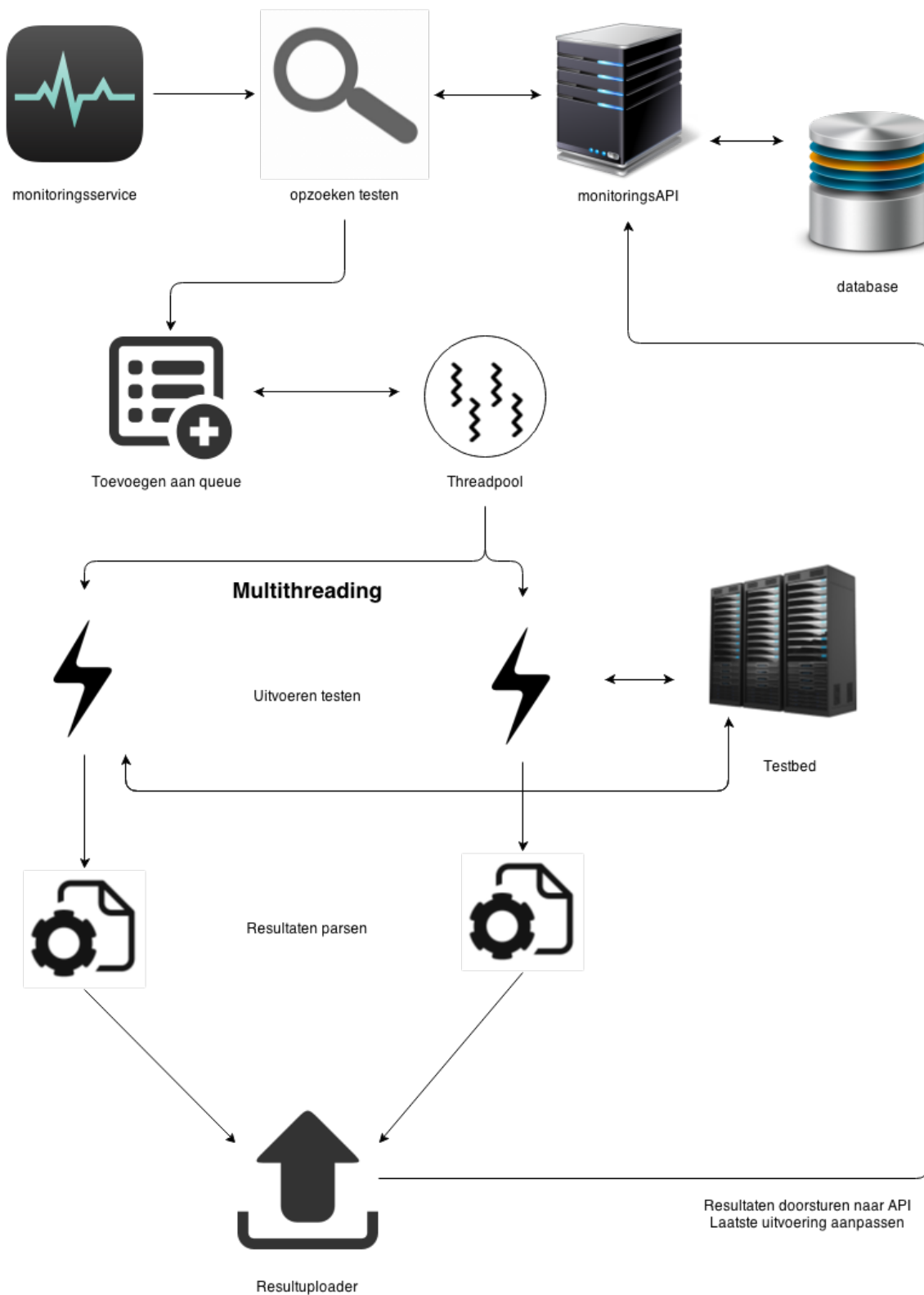
Figuur 5.6: Naast de console uitvoer zijn ook het originele resultaat in XML formaat en het overeenkomstige HTML formaat beschikbaar.

5.4 Service

The service werkt met meerdere threads. Er is één hoofdthread die testen ophaalt via de API en deze toevoegt aan een queue. Vervolgens worden de testen één voor één uit de queue gehaald en uitgevoerd door de threadpool. De grootte van de threadpool is instelbaar. Standaard is dit $cpu_{cores} * 2 + 1$.

Elke test wordt uitgevoerd op een aparte thread. Tijdens de uitvoering wordt verbinding gemaakt met één of meerdere testbeds via de jFed automated tester. Deze geeft een XML-file terug waaruit de tussenresultaten geparset worden. Deze resultaten worden vervolgens doorgegeven aan de resultuploader.

De resultuploader staat in voor het doorsturen van resultaten naar de API en draait op een aparte thread. Hierbij wordt ook de volgende uitvoeringstijd aangepast. Dit proces wordt beschreven in Figuur 5.7.



Figuur 5.7: De werking van de monitoringsservice

5.5 Loadtest

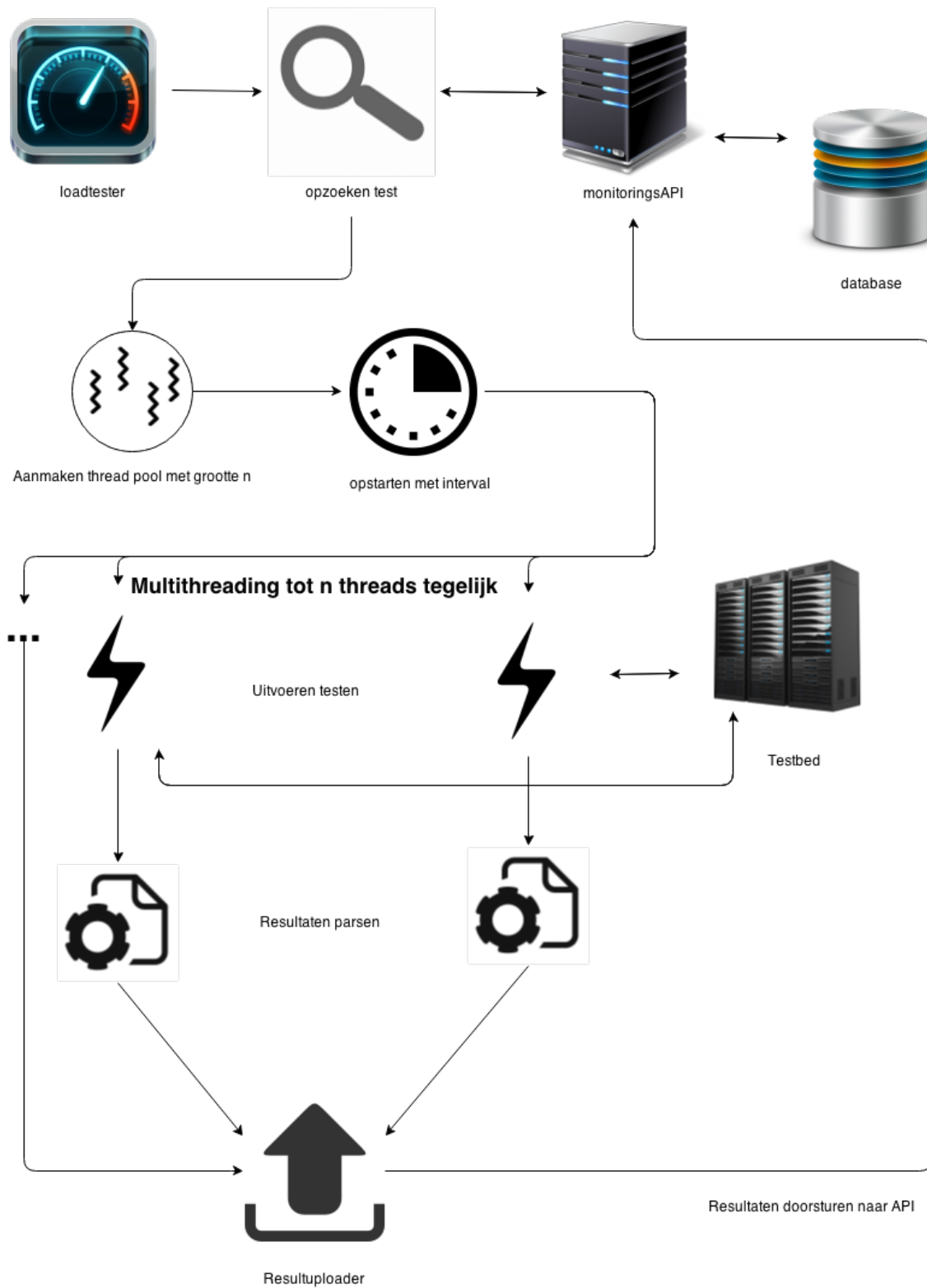
5.5.1 Uitwerking

De uitwerking wordt weergegeven in Figuur 5.8. Let op de overeenkomsten met Figuur 5.7, waar de monitoringsservice wordt uitgelegd. De uitwerking van de loadtest is op een aantal stappen na, gelijk aan de uitwerking van de monitoringsservice, vermits de kern van beide gelijk is.

Eerst wordt de bestaande test opgehaald. Daarna wordt een threadpool aangemaakt met de grootte n ; het aantal testen dat uitgevoerd moet worden. Hierdoor kunnen alle testen tegelijkertijd uitgevoerd worden. Vervolgens wordt elke test opgestart na een instelbaar interval. Dit interval zorgt er bijvoorbeeld voor dat de testen elkaar om de 2 seconden opvolgen.

Eenmaal een test is opgestart en uitgevoerd, wordt elk resultaat geparset en doorgegeven aan de resultuploader. Deze zal de resultaten één voor één uploaden en daarbij de uitvoeringstijd van de volgende test niet aanpassen. Dat laatste zou ertoe leiden dat volgende resultaten niet meer aanvaard worden. Merk het verschil met de monitoringsservice die hier wel een aanpassing doet van het nextrun veld, zie ook bijlage B. Dit veld houdt bij wanneer de test een volgende keer uitgevoerd wordt.

Na het uitvoeren van de testen zijn de resultaten van de stresstest beschikbaar via de monitoringsAPI. De huidige implementatie maakt geen onderscheid tussen resultaten van de monitoringsservice en de loadtesten. In plaats daarvan wordt de databank samen met de API gekloond. De resultaten zijn beschikbaar in de gekloonde databank, die na elke stresstest gewist wordt om de resultaten overzichtelijk te houden. Samenvoegen van de monitoringresultaten met de loadtestresultaten is mogelijk, maar bleek noch een prioriteit, noch noodzakelijk te zijn.



Figuur 5.8: De uitwerking van de loadtesten.

5.5.2 Voorbeeld van een stresstest

Deze sectie bevat een voorbeeld van een uitgevoerde loadtest als voorbereiding voor een practicum door de Griekse universiteit van Patras. Tijdens dit practicum zullen een 50-tal studenten elk 2 PC's gebruiken om TCP congestion testen. Hierbij zal de virtual wall of wall1 gebruikt worden als testbed. Doordat alle studenten hun experiment op zeer korte tijd van mekaar zullen starten, zal een grote belasting ontstaan op het testbed. Om te kijken of het testbed dergelijke belasting aankan, wordt een loadtest van 119 PC's uitgevoerd. Deze loadtest zal het practicum simuleren en weergeven of er problemen optreden. De loadtest is uitgevoerd met de code die geschreven werd in de masterproef. Het meten van resultaten en weergeven van grafieken is gebeurd door andere tools.

Figuur 5.9 geeft een overzicht van de gebruikte computers. Deze computers zijn per twee opgedeeld in slices. Figuur 5.10 geeft een overzicht van deze slices met bijhorende status. Op deze figuur komt een groen vak overeen met een slice die gelukt is, een rood vak daarentegen duidt op problemen.

Project wall2-ilabt-iminds-be is using 119 PCs!

Experiments	Groups	Members	Profile	Project Stats
Project Experiments				
EID	State	Nodes [1]	Hours Idle [2]	Description
221111a	active	2	0	urn:publicid:IDN+wall2.ilabt.iminds.be:patras+slice+221111a
228004a	active	2	0	urn:publicid:IDN+wall2.ilabt.iminds.be:patras+slice+228004a
228005a	active	2	0	urn:publicid:IDN+wall2.ilabt.iminds.be:patras+slice+228005a
228009a	active	2	0	urn:publicid:IDN+wall2.ilabt.iminds.be:patras+slice+228009a
228015a	active	2	0	urn:publicid:IDN+wall2.ilabt.iminds.be:patras+slice+228015a
228016a	active	2	0	urn:publicid:IDN+wall2.ilabt.iminds.be:patras+slice+228016a
228024a	active	2	0	urn:publicid:IDN+wall2.ilabt.iminds.be:patras+slice+228024a
228026a	active	2	0.08	urn:publicid:IDN+wall2.ilabt.iminds.be:patras+slice+228026a
228030a	active	2	0	urn:publicid:IDN+wall2.ilabt.iminds.be:patras+slice+228030a
228033a	active	2	0	urn:publicid:IDN+wall2.ilabt.iminds.be:patras+slice+228033a
228044a	active	2	0	urn:publicid:IDN+wall2.ilabt.iminds.be:patras+slice+228044a
228045a	active	2	0	urn:publicid:IDN+wall2.ilabt.iminds.be:patras+slice+228045a
228050a	active	2	0	urn:publicid:IDN+wall2.ilabt.iminds.be:patras+slice+228050a
228069a	active	2	0	urn:publicid:IDN+wall2.ilabt.iminds.be:patras+slice+228069a
228079a	active	2	0	urn:publicid:IDN+wall2.ilabt.iminds.be:patras+slice+228079a
228086a	active	2	0	urn:publicid:IDN+wall2.ilabt.iminds.be:patras+slice+228086a
228100a	active	2	0	urn:publicid:IDN+wall2.ilabt.iminds.be:patras+slice+228100a
228103a	active	2	0	urn:publicid:IDN+wall2.ilabt.iminds.be:patras+slice+228103a
228109a	active	2	0	urn:publicid:IDN+wall2.ilabt.iminds.be:patras+slice+228109a
228110a	active	2	0	urn:publicid:IDN+wall2.ilabt.iminds.be:patras+slice+228110a
228122a	active	2	0	urn:publicid:IDN+wall2.ilabt.iminds.be:patras+slice+228122a
228130a	active	2	0	urn:publicid:IDN+wall2.ilabt.iminds.be:patras+slice+228130a
228138a	active	2	0	urn:publicid:IDN+wall2.ilabt.iminds.be:patras+slice+228138a
228145a	active	2	0	urn:publicid:IDN+wall2.ilabt.iminds.be:patras+slice+228145a
228149a	active	2	0	urn:publicid:IDN+wall2.ilabt.iminds.be:patras+slice+228149a
228153a	active	2	0	urn:publicid:IDN+wall2.ilabt.iminds.be:patras+slice+228153a
228155a	active	2	0	urn:publicid:IDN+wall2.ilabt.iminds.be:patras+slice+228155a
228156a	active	2	0.09	urn:publicid:IDN+wall2.ilabt.iminds.be:patras+slice+228156a
228161a	active	2	0	urn:publicid:IDN+wall2.ilabt.iminds.be:patras+slice+228161a
228162a	active	2	0	urn:publicid:IDN+wall2.ilabt.iminds.be:patras+slice+228162a
228168a	active	2	0	urn:publicid:IDN+wall2.ilabt.iminds.be:patras+slice+228168a
228172a	active	2	0	urn:publicid:IDN+wall2.ilabt.iminds.be:patras+slice+228172a
228174a	active	2	0	urn:publicid:IDN+wall2.ilabt.iminds.be:patras+slice+228174a
228176a	active	2	0	urn:publicid:IDN+wall2.ilabt.iminds.be:patras+slice+228176a
228184a	active	2	0.09	urn:publicid:IDN+wall2.ilabt.iminds.be:patras+slice+228184a
228185a	active	2	0	urn:publicid:IDN+wall2.ilabt.iminds.be:patras+slice+228185a
228188a	active	2	0	urn:publicid:IDN+wall2.ilabt.iminds.be:patras+slice+228188a
228192a	active	2	0	urn:publicid:IDN+wall2.ilabt.iminds.be:patras+slice+228192a
228197a	active	2	0	urn:publicid:IDN+wall2.ilabt.iminds.be:patras+slice+228197a
228203a	active	2	0	urn:publicid:IDN+wall2.ilabt.iminds.be:patras+slice+228203a
228211a	active	2	0	urn:publicid:IDN+wall2.ilabt.iminds.be:patras+slice+228211a
228217a	active	2	0	urn:publicid:IDN+wall2.ilabt.iminds.be:patras+slice+228217a
228220a	active	2	0	urn:publicid:IDN+wall2.ilabt.iminds.be:patras+slice+228220a
228222a	active	2	0	urn:publicid:IDN+wall2.ilabt.iminds.be:patras+slice+228222a
228223a	active	2	0	urn:publicid:IDN+wall2.ilabt.iminds.be:patras+slice+228223a

Figuur 5.9: Overzicht van de verschillende slices.



Figuur 5.10: Status van de slices.

Vermits er in Figuur 5.10 maar één pc rood is, kan besloten worden dat het testbed dergelijke belasting kan verwerken. Figuur 5.11 hieronder geeft een grafiek van de belasting van het testbed op die dag. Uiterst rechts op de figuur zijn er 2 hoge waarden zichtbaar, deze zijn veroorzaakt door de stresstest. Doordat het testbed een dubbel aantal cores ziet dan dat er werkelijk zijn, moeten de percentages verdubbeld worden. De stresstest met 119 computers geeft bijgevolg een belasting van 80% voor een bepaalde tijd.



Figuur 5.11: Belasting van het testbed, percentages moeten verdubbeld worden.

Vervolgens werd de test herhaald met 100 gebruikers die telkens 2 computers gebruiken. De veroorzaakte belasting is te zien in Figuur 5.12. Ook hier moeten de percentages verdubbeld worden wat een belasting van 90% geeft over een langere periode.



Figuur 5.12: Belasting van een testbed met 100 gebruikers

De test met 100 gebruikers bestaat eigenlijk uit 100 testen die tegelijk draaien. Voor elke test is bijgevolg een resultaat, dat te zien is op de monitoringsinterface⁵. Figuur 5.13 geeft resultaten per test voor 100 users. Het is duidelijk dat er hier en daar problemen optreden, maar dat het overgrote deel wel slaagt.

Doordat de load veel hoger is dan wat de studenten in werkelijkheid zullen doen is de stresstest succesvol.

⁵Deze monitoringsinterface is een kloon van de werkende monitoringsAPI die enkel stresstesten uitvoert.

03/06/2014 - 22:04:59	16:28	0	1	2	3	4	5	6	7	8	log	resultsHtml	overview
03/06/2014 - 22:04:58	19:04	0	1	2	3	4	5	6	7	8	log	resultsHtml	overview
03/06/2014 - 22:04:53	17:59	0	1	2	3	4	5	6	7	8	log	resultsHtml	overview
03/06/2014 - 22:04:48	18:25	0	1	2	3	4	5	6	7	8	log	resultsHtml	overview
03/06/2014 - 22:03:39	17:34	0	1	2	3	4	5	6	7	8	log	resultsHtml	overview
03/06/2014 - 22:03:39	18:26	0	1	2	3	4	5	6	7	8	log	resultsHtml	overview
03/06/2014 - 22:03:39	17:41	0	1	2	3	4	5	6	7	8	log	resultsHtml	overview
03/06/2014 - 22:03:39	17:19	0	1	2	3	4	5	6	7	8	log	resultsHtml	overview
03/06/2014 - 22:03:39	18:38	0	1	2	3	4	5	6	7	8	log	resultsHtml	overview
03/06/2014 - 22:03:38	16:58	0	1	2	3	4	5	6	7	8	log	resultsHtml	overview
03/06/2014 - 22:03:38	16:04	0	1	2	3	4	5	6	7	8	log	resultsHtml	overview
03/06/2014 - 22:03:37	16:51	0	1	2	3	4	5	6	7	8	log	resultsHtml	overview
03/06/2014 - 22:03:37	17:03	0	1	2	3	4	5	6	7	8	log	resultsHtml	overview
03/06/2014 - 22:03:36	16:40	0	1	2	3	4	5	6	7	8	log	resultsHtml	overview
03/06/2014 - 22:03:36	16:48	0	1	2	3	4	5	6	7	8	log	resultsHtml	overview
03/06/2014 - 22:03:36	17:47	0	1	2	3	4	5	6	7	8	log	resultsHtml	overview
03/06/2014 - 22:03:36	18:32	0	1	2	3	4	5	6	7	8	log	resultsHtml	overview
03/06/2014 - 22:03:34	16:24	0	1	2	3	4	5	6	7	8	log	resultsHtml	overview
03/06/2014 - 22:03:34	17:18	0	1	2	3	4	5	6	7	8	log	resultsHtml	overview
03/06/2014 - 22:03:34	18:04	0	1	2	3	4	5	6	7	8	log	resultsHtml	overview
03/06/2014 - 22:03:29	17:07	0	1	2	3	4	5	6	7	8	log	resultsHtml	overview
03/06/2014 - 22:03:22	15:46	0	1	2	3	4	5	6	7	8	log	resultsHtml	overview
03/06/2014 - 22:03:18	16:19	0	1	2	3	4	5	6	7	8	log	resultsHtml	overview
03/06/2014 - 22:03:16	15:34	0	1	2	3	4	5	6	7	8	log	resultsHtml	overview
03/06/2014 - 22:03:14	17:03	0	1	2	3	4	5	6	7	8	log	resultsHtml	overview
03/06/2014 - 22:01:38	15:03	0	1	2	3	4	5	6	7	8	log	resultsHtml	overview
03/06/2014 - 22:01:38	16:03	0	1	2	3	4	5	6	7	8	log	resultsHtml	overview
03/06/2014 - 22:01:37	16:29	0	1	2	3	4	5	6	7	8	log	resultsHtml	overview
03/06/2014 - 22:01:33	16:01	0	1	2	3	4	5	6	7	8	log	resultsHtml	overview
03/06/2014 - 22:01:16	15:55	0	1	2	3	4	5	6	7	8	log	resultsHtml	overview
03/06/2014 - 22:01:16	15:31	0	1	2	3	4	5	6	7	8	log	resultsHtml	overview
03/06/2014 - 22:01:15	15:49	0	1	2	3	4	5	6	7	8	log	resultsHtml	overview
03/06/2014 - 22:01:15	13:12	0	1	2	3	4	5	6	7	8	log	resultsHtml	overview
03/06/2014 - 22:01:12	14:45	0	1	2	3	4	5	6	7	8	log	resultsHtml	overview
03/06/2014 - 22:01:04	16:17	0	1	2	3	4	5	6	7	8	log	resultsHtml	overview
03/06/2014 - 22:00:38	15:33	0	1	2	3	4	5	6	7	8	log	resultsHtml	overview
03/06/2014 - 21:59:13	14:52	0	1	2	3	4	5	6	7	8	log	resultsHtml	overview
03/06/2014 - 21:59:13	14:55	0	1	2	3	4	5	6	7	8	log	resultsHtml	overview
03/06/2014 - 21:59:12	14:21	0	1	2	3	4	5	6	7	8	log	resultsHtml	overview
03/06/2014 - 21:59:12	15:40	0	1	2	3	4	5	6	7	8	log	resultsHtml	overview
03/06/2014 - 21:59:11	15:25	0	1	2	3	4	5	6	7	8	log	resultsHtml	overview
03/06/2014 - 21:59:11	14:18	0	1	2	3	4	5	6	7	8	log	resultsHtml	overview
03/06/2014 - 21:59:11	14:20	0	1	2	3	4	5	6	7	8	log	resultsHtml	overview

Figuur 5.13: Weergave per test

Appendix A

MonitoringAPI reference

A.1 About

This chapter will provide a detailed list of methods and functions available in the monitoringAPI. The monitoringAPI provides access to all the monitoringdata, both results and configuration. The API is available at <http://f4f-mon-dev.intec.ugent.be/service/index.php> , however this location is not permanent. In case the url returns 404, one should contact IBC-N/iMinds. Keep in mind that the API is still under development, so things might change.

The service supports get and post requests, both are handled in the same way unless stated otherwise. A request should use <http://f4f-mon-dev.intec.ugent.be/service/index.php/> as base url e.g. <http://f4f-mon-dev.intec.ugent.be/service/index.php/<functionname>?<arguments>>¹.

A.2 Introduction

The system contains all information about the results, testbeds, testdefinitions, testinstances and the login information of users used for testing. It is important to note that these users are used by programs and are not logins for experimenters.

A testdefinition is a definition of the test e.g. a ping test consists of a ping command while a login test uses the Aggregate manager to log in to a testbed. The definition describes the arguments used for and results returned from a test. Each definition has its own name, the testdefinitionname and an internal testtype named testtype this type tells the monitoringsservice whether or not to use the automated tester.

The testinstance on the other hand is an instance of a testdefinition e.g. a login test on the virtual wall. Each instance has an instanceid, a name and the name of its definition.

¹Of course the < and > shouldn't be in the actual url.

A.3 Functions

The functions needed for experimenters are listed below. Next up is a detailed explanation of each function. The current list of functions is:

1. List
2. Last
3. TestDefinition
4. TestInstance
5. Testbed
6. User
7. Q (GENI datastore support, use with caution)
8. AddResult (administrators only)
9. updateNextRun (administrators only)

The return format of each call below can be change with `?format=<formatname>`. Other than JSON as default format, PrettyJson is also available. PrettyJson uses the PHP JSON pretty print parameter to outline the resulting JSON making it more humanfriendly. Use it by adding `format=PrettyJson` to any call.

It is also possible to use multiple values by using a comma separated list e.g. `...?testbed=wall1,wall2` will filter on testbed where the testbed is either wall1 or wall2.

A.3.1 List

What

The list function will list monitoring results. Because of performance and security reasons, only the last 100 results that comply with the query will be returned. Keep in mind that this is 100 for each testinstance, testbed combination. In case more is needed, one could use from and till clauses or contact IBCN/iMinds.

Arguments

1. param: value of any parameter of the corresponding test. e.g. testbed, user, ...
2. testdefinitionname: the name of the testdefinition of the corresponding testbed.
3. resultid: the id of the result.
4. testname: the name of the corresponding test
5. testid: the testinstanceid of the corresponding test
6. from: search results after a given timestamp NOTE: use iso 8601 format WITH time-zone.
7. till: search results before a given timestamp NOTE: use iso 8601 format WITH timezone.
8. count: the last <count> results will be returned. Limited to 100 and default also 100. The results are counted for each combination of testbed en testdefinitionname, therefore list?count=50 will return the last 50 results for each test on each testbed, so when there are 2 testbeds, 100 results will be returned in total.

It is important to note here that using from/till arguments in combination with count is not possible. Doing so will return a http 400 (bad request) error.

Examples

```
list?from=2014-03-18T19:29:00&till=2014-03-19T18:29:00&testdefinitionname=ping
&param=wall2
```

Will return all the results of each² pingtest on wall2 between 18/03/2014 18:29:00 and 19/03/2014 18:29:00.

```
list?testdefinitionname=stitch&count=5&param=wall2,wall1
```

Will return results of the last 5 stitching results for wall2 and the last 5 stitching results for wall1.

²Although generally not recommended, it is possible to have multiple tests of the same type on one testbed.

A.3.2 Last

What

Last has the same functionality as list, but uses default count=1. It is a shortcut to get the last results of each test on each testbed.

Arguments

1. param: value of any parameter of the corresponding test. e.g. testbed, user, ...
2. testdefinitionname: the name of the testdefinition of the corresponding testbed.
3. resultid: the id of the result.
4. testname: the name of the corresponding test
5. testid: the testinstanceid of the corresponding test
6. count: the last <count> results will be returned. Limited to 100 and default also 100. The results are counted for each combination of testbed en testdefinitionname therefor list?count=50 will return the last 50 results for each test on each testbed, so when there are 2 testbeds, 100 results will be returned in total.

Since last uses a default count, it is not possible to use from/till arguments here. Doing so will return a http 400 (bad request) error.

Examples

```
last?testdefinitionname=ping
```

Will return the last result of each³ pingtest of each testbed.

```
last?testdefinitionname=stitch&count=5&param=wall2
```

Will return results of the last 5 stitching tests for wall2. Note that this is also possible with list.

```
last
```

Returns the last result of each test on each testbed.

³Although generally not recommended, it is possible to have multiple tests of the same type on one testbed.

A.3.3 TestDefinition

What

This call will return the testdefinitions. A testdefinition is used to define a test.

Arguments

1. testdefinitionname: the name of the testdefinition.
2. testtype: the internal testtype used to determine if the test uses the automated tester or a bash command. It is not recommended for experimenters to use this.

Examples

```
testdefinition?testdefinitionname=ping,login2
```

Will return the definition of the ping test and of the login (AMv2) test.

A.3.4 TestInstance

What

This call will return the testinstance. A testinstance is an instance of the definition; the instance will define a pingtest on a certain testbed while a definition will define that a pingtest consists of a ping command.

Arguments

1. param: value of any parameter of the corresponding test. e.g. testbed, user, ...
2. testdefinitionname: the name of the testdefinition.
3. testname: name of the test.
4. testinstanceid: the id of the testinstance.
5. nextrun: field used to determine the next execution of a test. formatted in iso 8601 WITH timestamps.

Examples

```
testinstance?testdefinitionname=stitch&testbed=wall2
```

Will return all the instances of the stitching type on wall2.

```
testinstance?nextrun=2014-06-19T12:00:00
```

Will return all tests with nextrun \geq 19/06/2014 12:00:00; used by the monitoringsservice.

```
testinstance?param=ftester
```

Will return all testinstances where the user is ftester.

A.3.5 Testbed

What

This call will return the testbeds.

Arguments

1. testbedname: name of the testbed.
2. urn: urn of the testbed.
3. url: url of the testbed.

Examples

```
testbed?urn=urn:publicid:IDN+wall1.ilabt.iminds.be+authority+cm
```

Will return the information of the testbed with urn =
urn:publicid:IDN+wall1.ilabt.iminds.be+authority+cm (wall1).

A.3.6 User

What

This call will return information about a user. Note that only users used by the monitoringsservice are stored here, making it impossible to see information about all users on the testbed.

Arguments

1. username: the name of the users.
2. userauthorityurn: the urn of the user used for authentication.

Examples

```
user?username=ftester
```

Will return the information for the ftester user.

A.3.7 Q**What**

This function is not real function but an argument of the list function. It is used to make this monitoringAPI compatible with the GENI datastore. However since both GENI and this API are in developpment, not every function is yet available.

Arguments

The arguments here are encoded as a json string, for more information i refer to <http://groups.geni.net/geni/wiki/OperationalMonitoring> and <http://groups.geni.net/geni/wiki/OperationalMonitoring/DatastorePolling>.

Not all of these functions are supported only these:

1. aggregate: it is possible to filter on aggregate name.
2. eventType: the evenname as defined by genidatastore.
3. ts: unix timestamp in micro(1/1000000) seconds.
 - (a) gte: greater than or equal.
 - (b) lt: lower than.

Example

```
list?q={
  filters:{
    eventType:[ops_monitoring:is_available],
    obj:{type:aggregate,id:[wall2,wall1]}
  }
}
```

Will return results of the is_available⁴ test for wall1 and wall2.

⁴The is_available is the GENI name for a getVersion test, so this call will return whether or not the getVersion test succeeded

Bijlage B

Onderhoud, beheer en gebruik

Dit hoofdstuk richt zich tot de systeembeheerder en bespreekt hoe het onderhoud van de de monitoring-API en service verloopt. Hierbij wordt de nadruk gelegd op het toevoegen, verwijderen, activeren en deactiveren van testen.

B.1 Over

Een testinstantie maakt de link tussen een testdefinitie en zijn set parameters. In volgende tekst wordt, tenzij anders vermeld, met test ook altijd een testinstantie bedoeld. Een test heeft altijd een type. Dit type wordt gedefinieerd door de testdefinition die ook opgeslagen zit in de databank. De bedoeling is dat een systeembeheerder eenvoudig nieuwe testtypes kan ondersteunen door nieuwe definities aan te maken. Dit kan op meerdere manier en is hieronder beschreven.

Daarnaast heeft een test ook een nextrun veld, dit veld bevat een iso 8601 timestamp met tijdzone en geeft aan wanneer de test de volgende keer uitgevoerd moet worden. De test zal uitgevoerd worden op het moment dat de monitoringsservice (die periodiek opgestart wordt via cron) de API polt naar testen die uitgevoerd moeten worden. Als nextrun dan in het verleden ligt, zal de test doorgestuurd worden en uitgevoerd worden. De nextrun zal aangepast worden door de monitoringsservice als volgt: $nextrun_{new} = starttime + frequency$.

Het is ook mogelijk om testen te deactiveren door het veld enabled op false te zetten. Deze testen zullen bijgevolg niet uitgevoerd worden ongeacht de waarde van hun nextrun veld. Vanzelfsprekend moet dit veld terug op true gezet worden om de test terug te activeren.

B.2 Toevoegen van testen (testinstancies)

Voor het toevoegen van een testinstanties moet er eerst gekeken worden naar het de bijhorende definitie. De definitie kan opgevraagd worden via `/testdefinition?testdefinitionname=<definitienaam>` (zie ook bijlage A). Vervolgens moeten alle parameters ingesteld worden.

Een voorbeeld, de definitie van een list resource test is:

```
"listResources": {
  "testdefinitionname": "listResources",
  "testtype": "listResources",
  "testcommand": "",
  "parameters": {
    "user": {
      "type": "user",
      "description": "user for authentication"
    }, "testbed": {
      "type": "testbed",
      "description": "testbed to get the list resources from"
    }, "context-file": {
      "type": "file",
      "description": "username = \n      passwordFilename = \n
      pemKeyAndCertFilename = \n      userAuthorityUrn = "
    }
  }, "returnValues": {
    "count": {
      "type": "int",
      "description": "free resources"
    }, "rspec": {
      "type": "file",
      "description": "path of rspec file"
    }
  }
}
```

Hier kan uit afgeleid worden dat een listResource test een user, testbed en context file nodig heeft om een count en rspec terug te geven. Parameters worden dan bijvoorbeeld met een python script ingevuld, zoals in het voorbeeld hieronder waarbij een listResources test voor wall2 gemaakt wordt.


```

import psycopg2 #postgresql databanken

#queries eerste geeft de testinstance id terug
addTestQ = "INSERT INTO testinstances (\
    testname,\
    testDefinitionName,\
    frequency,\
    nextrun,\
    enabled\
) VALUES(%s,%s,%s,%s,%s) RETURNING testinstanceid"

addParQ = "INSERT INTO parameterInstances (\
    testinstanceId,\
    parameterName,\
    parametervalue\
) VALUES (%s,%s,%s)"

#connectie maken
con = psycopg2.connect(database=dbname, user=user, password=pass, host=durl)
#vul overeenkomstige waarden in
cur = con.cursor()

#maak test
cur.execute(addTestQ,(\
    "wall2list",\
    "listResources",\
    3600,\
    "2014-6-1T12:00:00+00",\
    "t")\
)
testinstanceid = cur.fetchone()[0] #ophalen id

#parameters
cur.execute(addParQ, (testinstanceid, "testbed", wall2))
cur.execute(addParQ, (testinstanceid, "user", ftester))

con.commit() #commit

```

Hiervoor moet het testbed wall2, de user ftester en de testdefinitie voor listResources bestaan.

B.3 Toevoegen van types/definities (testdefinitions)

Definities kunnen op meerdere manieren toegevoegd worden.

1. nieuw intern type
2. testdefinitie toevoegen in de databank.

B.3.1 Nieuw intern type

Het is zo dat elke test een intern type heeft. Dit type komt overeen met een klasse in de `monitor.testCalls` package . Er wordt gebruik gemaakt van overerving waarbij er 2 groepen testen zijn: de bashtesten die een bash commando uitvoeren en de jfed automated testen die gebruik maken van de jfed automated tester. Voor de eerste groep is het aangeraden om over te erven van `BashTestCall`, voor de tweede groep is `javaMainTestCall` de klasse. Overerven is sterk aangeraden vermits deze code bevat voor het parsen van parameters en het aanroepen van de jfed automated tester. In deze overgeërfde klasse kunnen parameters vervolgens hardgecodeerd worden, bijgevolg moeten ze niet meer opgeslagen worden in de databank.

Op deze manier is het mogelijk om bijvoorbeeld een eenvoudige stitching test te maken met een aantal standaard waarden. Naast deze test zou er nog een advanced stitching test kunnen zijn waar de parameters wel opgegeven moeten worden. Beide testen zouden vervolgens verschillen in naam. Bijgevolg kan men eenvoudig kiezen tussen beide varianten door het overeenkomstige testtype te kiezen.

Merk op dat voor het gebruik van nieuwe parameter types er mogelijk aanpassingen gedaan moeten worden in de `javaMainTestCall` klasse. Daarnaast moet er ook een nieuwe casse toegevoegd worden bij de `makeTest` van de `TestCallFactory`.

Een voorbeeld voor een login test (amv2) is volgende interne klasse:

```
public class LoginTestCall extends JavaMainTestCall {
    ...
    @Override
    protected ArrayList<String> getParameters(String parsedCommand) {
        ArrayList<String> commands = super.getParameters(parsedCommand);
        commands.add("--test-class");
        commands.add(
            "be.iminds.ilabt.jfed.lowlevel.api.test.TestAggregateManager2"
        );
        commands.add("--group");
        commands.add("nologin");
        commands.add("--output-dir");
        commands.add(makeTestOutputDir());

        return commands;
    }
    ...
}
```

De parameters worden hier hardgecodeerd, bijgevolg moeten deze parameters niet meer opgegeven worden bij het toevoegen van de test. Zo is hieronder een voorbeeld van python code die een instantie van bovenstaande test toevoegt, waarbij enkel een testbed en een user opgegeven moet worden. Andere parameters voor de automated tester zoals de test-class en group zijn hier niet meer nodig.

```
cur.execute(addTestQ, ("wall2login2", "login2", loginFreq, nextRun, enabled))
testinstanceid = cur.fetchone()[0]
cur.execute(addParQ, (testinstanceid, "testbed", map['testbedname']))
cur.execute(addParQ, (testinstanceid, "user", map['username']))
```

Er moet nog altijd een definitie voor login2 aanwezig zijn in de databank. Deze definitie zal de hardgecodeerde parameters echter niet meer bevatten. Toch is het noodzakelijk dat deze definitie aanwezig is, vermits ze o.a. gebruikt wordt om te kijken of alle tussenresultaten gegeven zijn wanneer er een nieuw resultaat toegevoegd moet worden.

B.3.2 Testdefinitie toevoegen in de databank

Een tweede manier is het toevoegen van een definitie in de databank, deze test heeft dan als intern type ofwel bash ofwel automatedTester. Bij de eerste wordt de commando tekst beschouwd als het commando, bij de tweede zijn dit de parameters die meegegeven worden aan de automated tester. Waarden van parameters kunnen opgevraagd worden door de parameter naam tussen < en > te plaatsen¹.

Op deze manier kan het voorgaande script omgezet worden naar een automatedTestCall waarbij de parameters in de databank opgeslagen zitten. Hieronder een voorbeeld in PHP:

```
$subQuery = "insert into parameterDefinitions (
    testDefinitionName ,
    parameterName ,
    parameterType ,
    parameterDescription
) values ($1,$2,$3,$4);";
$query = "insert into testdefinitions(
    testDefinitionName ,
    testtype ,
    geniDatastoreTestname ,
    geniDatastoredesc ,
    geniDatastoreUnits ,
    testcommand
) values ($1,$2,$3,$4,$5,$6);";
```

¹Dit is overal zo, alle waarden tussen < en > zullen geparset worden en opgezocht worden bij de parameters, indien het om een parameter met type testbed en naam testbed gaat geeft <testbed.urn> de urn van het testbed terug.

```

//test toevoegen
$data = array('loginGen','automatedTester',' ',' ','boolean',
  '--context-file <context-file> '
  . '--test-class '
  . 'be.iminds.ilabt.jfed.lowlevel.api.test.TestAggregateManager2 '
  . '--group nodelogin '
  . '--output-dir <output-dir> '
  . '-q'
  );
pg-query-params($con, $query, $data);

//parameters
$data = array("loginGen", "context-file", "file", "username = ftester
  passwordFilename = " . $authDir . $passfile . "
  pemKeyAndCertFilename = " . $authDir . "cert.pem
  userAuthorityUrn = <userAuthorityUrn>
  testedAggregateManagerUrn = <testedAggregateManager.urn>");
pg-query-params($con, $subQuery, $data);
...

```

De parameters zijn hier wel zichtbaar in de defintie. Tot slot kan vermeld worden dat `<context-file>` vervangen zal worden door de parameter `context-file`. Vermits deze parameter van het type `file` is, wordt er een file aangemaakt waarvan het path terecht komt op de plaats van de `<context-file>`.

B.3.3 User toevoegen

Een user wordt, binnen deze context, gebruikt door de jFed automated tester om de authenticatie op een testbed af te handelen. Om flexibiliteit te verhogen is het mogelijk om meerdere users aan te maken. Deze kunnen vervolgens elk gebruikt kunnen worden voor de authenticatie op een testbed.

Een user heeft een username, een certificaat en een geassocieerde urn. Het certificaat is mogelijk geëncrypteerd en zit niet in de databank, maar in een aparte file opgeslagen. Deze file is terug te vinden in `/ssl/<pemKeyAndCertFilename>`. Indien het certificaat geëncrypteerd is, is het bijhorende wachtwoord opgeslagen in de `passwordFilename`. Deze file komt overeen met `/ssl/<passwordFilename>`.

Een user kan bijvoorbeeld met volgende query in python toegevoegd worden:

```
addUserQ = "INSERT INTO users (username , \
        userAuthorityUrn , \
        passwordFilename , \
        pemKeyAndCertFilename) VALUES(%s,%s,%s,%s)"
```

Hierbij moet het certificaat opgeslagen worden in `/ssl/`. Het certificaat wordt apart opgeslagen omdat jFed een path nodig heeft en niet het certificaat zelf. Daarnaast zorgt de scheiding hiervan ook dat het opvragen van users niet beveiligd moet worden, vermits het certificaat niet zichtbaar is.

Testbed toevoegen

Een testbed heeft een naam, een urn en een url die gebruikt om de pinglatency te bepalen. Deze query in python kan gebruikt worden om een testbed toe te voegen.

```
addBedQ = "INSERT INTO testbeds (testbedname , url , urn) VALUES(%s,%s,%s)"
```

B.4 Troubleshooting

B.4.1 Resultaten worden niet opgeslagen door de API

Resultaten kunnen om één van volgende redenen geweigerd worden:

1. het testid bestaat niet of is onjuist
2. Niet alle tussenresultaten die in de testdefinitie gedefinieerd zijn, zijn meegegeven.
3. nextrun ligt niet na de bestaande nextrun. Dit kan ontstaan door fouten met tijdzones. Normaal gezien wordt een unix timestamp gebruikt in microseconden (1/1000 000 seconde).

De regel is ook dat ofwel alles wordt opgeslagen ofwel niets. Er wordt gebruik gemaakt van transacties die de integriteit van de data bewaren.

B.4.2 Het programma loopt vast tijdens het uitvoeren van een zelf aange maakte testen.

Controleer of alle parameter namen, opgegeven in de instance, overeenkomen met de namen van de parameters in de definitie.

Bibliografie

- FIRE (2014). What is fire. URL <http://www.ict-fire.eu/getting-started/what-is-fire.html>.
- GENI (2014a). About geni. URL https://www.geni.net/?page_id=2.
- GENI (2014b). Geni operational monitoring project. URL <http://groups.geni.net/geni/wiki/OperationalMonitoring>.
- GENI (2014c). Overview of operational monitoring. URL <http://groups.geni.net/geni/wiki/OperationalMonitoring/Overview>.
- GENI (2014d). Resource specification (rspec). URL <http://groups.geni.net/geni/wiki/GENIConcepts#TheGENIAMAPIandGENIRSpecs>.
- iMinds (2014a). Federation for fire. URL <http://www.iminds.be/nl/projecten/2014/03/07/fed4fire>.
- iMinds (2014b). jfed : Java based framework to support sfa testbed federation client tools. URL <http://jfed.iminds.be/>.
- iMinds (2014c). Over iminds. URL <http://www.iminds.be/nl/over-ons>.
- L. Peterson, R. Ricci, A. Falk & J. Chase (2010). Slice-based federation architecture. URL <http://groups.geni.net/geni/attachment/wiki/SliceFedArch/SFA2.0.pdf>.
- B. Vermeulen (2014). D2.4 - second federation architecture. URL http://www.fed4fire.eu/fileadmin/documents/public_deliverables/D2-4_Second_federation_architecture_Fed4FIRE_318389.pdf.
- T. Walcarius, W. Vandenberghe, B. Vermeulen, P. Demeester & D. Davies (2014). Health monitoring of federated future internet experimentation facilities. In *European Conference on Networks and Communications 2014 (EuCNC 2014)*. Bologna, Italy.