UNIVERSITEIT GENT

Academiejaar 2013–2014

Faculteit Ingenieurswetenschappen en Architectuur
Valentin Vaerwyckweg 1 – 9000 Gent

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. Dit samenwerkingsakkoord zal trachten om de uitwisseling van ideeën tussen onderzoekers te verhogen.

Daarnaast wordt, door de ontwikkeling van een gemeenschappelijke architectuur, het delen van testfaciliteiten gemakkelijker gemaakt genaamd SFA (Slice Federation Architecture). Om het leven van onderzoekers gemakkelijker te maken is jFed ontworpen. jFed wordt gebruikt om testfaciliteiten aan te sturen via SFA. Deze manier van werken zorgt ervoor dat er snel proefopstellingen op verschillende testfaciliteiten. Toch zijn er ook enkele nadelen verbonden aan deze manier van werken. Het is voor een onderzoeker soms erg moeilijk om te bepalen of een bepaald gedrag in zijn experiment te wijten is aan eigen ontwikkelingen, of aan het falen van een testbed.

Deze masterproef verhelpt dit probleem door de invoer van een automatisch monitoringsproces. Een monitoringsservice zal instaan voor de monitoring van de testfaciliteiten. Hierbij wordt informatie verzameld die via een monitoringsAPI ter beschikking gesteld wordt. Deze API (application programming interface) vormt een stevige basis waarop andere toepassingen kunnen gemaakt worden.

Dit monitoringssysteem wordt ook omgebouwd om loadtesten te kunnen uitvoeren waarbij gekeken wordt 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.

To make handling of these different testbeds easier, jFed was created. jFed is a java tool with the purpose of controlling testbeds. jFed uses the SFA (Slice Federation Architecture) to control and configure testbeds. Unfortunately, the current situation has a major downside in that it is very hard for researchers to determine if a certain behavior is caused by the testconfiguration or by the test facility.

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

Inhoudsopgave

Inleiding

Het gebruik van netwerken en het internet om computers en allehande randapperatuur 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 onderzoeksproject dat zich op deze doelen richt.

Om de configuratie en werking van de verschillende testbeds gelijk te trekken, 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 de SFA 2.0 (Slice-federation-architecture). SFA deelt een testbed op in meerdere niveau's, het onderste niveau zijn de componenten. Een component is een computer, router, ... van een testbed. Een term die hier ook vaak komt bij kijken is een node. Een node is een computer binnen een testbed. Een node is dus een component van een bepaald type.

Omdat testbeds vaak door meerdere onderzoekers tegelijk gebruikt worden, worden componenten vaak gemultiplexed. Zo krijgt elke onderzoeker 'een stuk' van een computer, een sliver. Meerdere slivers vormen samen een slice. Een slice is dus een verzameling componenten waarop een experiment gedraaid wordt.

Daarnaast kan men de componenten ook groeperen in aggregaten. Een aggregate is een verzameling componenten die valt onder eenzelfde beheerder. Een goed voorbeeld van een aggregate is de virtual wall. De virtual wall is een testfaciliteit van iMinds, gebruikt om netwerken te simuleren. De virtual wall is een aggregate, in die zin dat alle componenten beheerd worden door iMinds. SFA wordt later in de scriptie uitgebreid besproken.

Aggregates die een vertrouwensrelatie hebben vormen een federatie. Fed4FIRE zal er dus voor zorgen dat alle testbeds van FIRE een federatie vormen. Hierdoor worden onderzoekers binnen FIRE vertrouwd op alle testbeds binnen FIRE.

Het beheer van al deze verschillende aggregates is geen sinecure. Om dit beheer te vereenvoudigen heeft iMinds binnen het Fed4FIRE project een monitoringsservice gemaakt. Deze service is echter snel ontwikkeld en is niet geschikt voor uitbereidingen. Deze masterproef bestaat uit 2 grote delen. Het eerste deel is een nieuwe monitoringsservice maken die de testbeds controleert. De informatie die deze service verzamelt zal beschikbaar gemaakt worden door een monitoringsAPI.

Naast het monitoren van betrouwbaarheid is het ook belangrijk om te weten welke belasting een testbed kan afhandelen. Het testen hiervan gebeurd met loadtesten of stresstesten. Hierbij wordt een testen meerdere keren opgestart in een heel kort tijdsinterval. Vervolgens wordt gekeken hoe het testbed reageert.

FIRE werkt samen met een gelijkaardig project, GENI. GENI (Global Environment for Network Innovations) is een Amerikaans project met gelijkaardige doelstellingen als FIRE. Hierdoor is de samenwerking tussen beide projecten zeer hoog. Het tweede deel van de masterproef is de integratie van monitoringsAPI in het GENI project.

Hoofdstuk 1 Situeert de masterproef. Hier wordt ook kort de opdracht uitgelegd.

Hoofdstuk 2 gaat dieper in op de SFA-architectuur. De SFA-architectuur wordt gebruikt om de configuratie en besturing van testbeds over heel de wereld gelijk te maken.

Hoofdstuk 3 maakt een analyse van de bestaande FIRE en GENI monitor en geeft hierbij ook de probleemstelling aan.

Hoofdstuk 4 geeft eerst de opbouw van de verschillende projecten binnen de masterproef. Vervolgens worden de gebruikte technologieën besproken.

Hoofdstuk 5 bespreekt per onderdeel de verschillende gebruikte technologieën.

Hoofdstuk 6 gaat dieper in de op de uitwerking. Hierbij wordt de grote structuur van de verschillende delen besproken.

Hoofdstuk 7 geeft uitleg over de uitwerking van de loadtesten.

Bijlage A is een concreet voorbeeld van een loadtest.

Bijlage A bevat de referentie van de monitoringsAPI, geschreven in Engels.

Bijlage Bt bevat informatie voor de systeembeheerder.

Bijlages A en B zijn specifiek gericht aan de systeembeheerder en eindgebruikers.

Hoofdstuk 1

Situering

In dit hoofdstuk wordt het achterliggende kader van de masterproef geschetst. Daarnaast wordt ook de opdracht uitgewerkt. De opdracht bestaat uit 2 grote delen. Het eerste deel is een monitoringsservice maken om testbeds te controleren. De informatie van deze monitoringsservice wordt via de monitoringsAPI beschikbaar gesteld aan de buitenwereld. Het tweede deel is de integratie van de monitoringsAPI in een groter Amerikaans framework.

1.1 Situering

Deze masterproef is een onderdeel van een groter Europees onderzoeksproject genaamd FIRE (Future Internet Research and Experimentation). FIRE is gericht is op onderzoek naar toekomstige internet- en netwerktechnologieën. Door onderzoekscentra te laten samenwerken(FIRE, 2014), tracht FIRE het onderzoek vlotter te laten verlopen. FIRE heeft twee grote doelen: enerzijds de samenwerking tussen verschillende onderzoekscentra verbeteren, anderzijds het delen van testfacilities gemakkelijker maken.

Het eerste doel is de samenwerking tussen verschillende onderzoekscentra te 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 te 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 hardware als software die gebruikt wordt om testen te verrichten. Een concreet voorbeeld van een testfaciliteit is een testbed. Een 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 cliënts 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 verloopt aan de hand van testbeds.

Het probleem dat zich hierbij stelt is dat elk testbed op zijn eigen manier werkt. Onderzoekers hebben nu wel toegang tot andere testbeds, maar moeten voor elk testbed eerst de nieuwe configuratie leren. Verschillende testbeds laten samenwerken op deze manier is geen sinecure. Om deze configuratie gelijk 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. Binnen een federatie worden alle onderzoekers, services en testbeds vertrouwd. Een tweede doel van deze architectuur is de configuratie en werking van testbeds gelijk te maken.

SFA 2.0 werkt met 3 niveaus voor verantwoordelijkheid. De bovenste is de MA (Management Authority), deze is verantwoordelijk voor de stabiliteit van een volledige testfaciliteit. Een SA (slice authority) is verantwoordelijk voor één of meerdere slices. De laatste is een gebruiker, bijvoorbeeld een onderzoeker die een experiment wil uitvoeren op een testbed.

Daarnaast maakt SFA ook gebruik van een specifieke naamgeving. Een component is een primaire block van de architectuur, bijvoorbeeld een computer of een router. Meerdere componenten worden vervolgens gegroepeerd in aggregaten. Alle componenten van een aggregaat vallen onder dezelfde MA (Management Authority). Elke aggregaat wordt gecontroleerd door een AM (aggregate manager). De AM beheert de allocatie van de verschillende experimenten op de aggregaat.

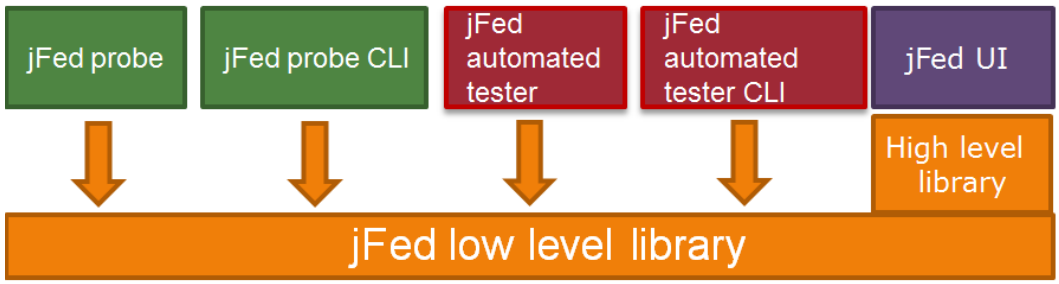
De MA (Management Authority) bepaalt vervolgens hoe de resources verdeeld worden. Indien een component gemultiplexed wordt spreekt van men slivers. De gebruiker heeft dan een sliver van de component tot zijn beschikking. Meerdere slivers worden gegroepeerd tot een slice. Een experiment wordt uitgevoerd binnen een slice. Meer uitleg over SFA architectuur volgt later in de scriptie.

Om onderzoek gemakkelijker te maken is er de tool jFed. jFed werd door iMinds ontwikkeld(iMinds, 2014b) en is een javatool die de SFA architectuur gebruikt 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.

Om dit probleem op te lossen heeft iMinds een monitoringssysteem uitgebouwd(Vermeulen, 2014). Dit monitoringssysteem werkt, maar is door de snelle ontwikkeling niet voorzien op uitbereidingen. Deze masterproef zal enerzijds een monitoringsservice maken die deze testbeds in de gaten houdt. Anderzijds zal deze informatie via een monitoringsAPI beschikbaar gemaakt worden voor onderzoekers. Het is de bedoeling dat deze API een stevige basis vormt waarop andere applicaties kunnen gebouwd worden. Merk op dat monitoring op 3 niveau's mogelijk is: component,slice en aggregate. De monitoringsservice die hier besproken wordt, richt zich op de bovenste laag. Deze laag kijkt of een testbed online is en hoeveel resources er beschikbaar zijn. Deze informatie is momenteel beschikbaar via een aantal websites, maar zit nog niet in de primaire gebruikersinterface.

De monitoringsservice zal gebruik maken van een module van jFed. jFed bestaat immers uit verschillende modules (Figuur 1.1). De jFed low level library zorgt samen met de high level library voor o.a. de beveiliging van verbindingen en het omzetten van argumenten naar de juiste codering. De jFed UI is de userinterface. De probe wordt hier buiten beschouwing gelaten. De jFed automated tester en automated tester CLI zijn, in het kader van deze masterproef, wel belangrijk. Deze zorgen immers voor het uitvoeren van monitoringstesten.



Figuur 1.1: Opbouw van jFed.

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 aggregates te bundelen en beschikbaar te stellen aan onderzoekers(GENI, 2014a). GENI maakt, net als FIRE, gebruik van de SFA architectuur(GENI, 2014b). Hierdoor is het mogelijk om onderzoekers van FIRE te laten werken op testbeds van GENI en omgekeerd.

GENI heeft zelf ook een gedistribueerde monitoringservice uitgebouwd(GENI, 2014c). Deze service maakt gebruik van datastores(GENI, 2014d). Een datastore houdt de monitoring informatie van een testbed of aggregate bij. Deze informatie wordt dan opgehaald door een collector. De webservice van FED4FIRE zou ook als een datastore bekeken worden. Op deze manier kan de monitoringsAPI geïntegreerd worden in een groter monitoringsframework. Dit zal als tweede deel van de masterproef behandeld worden. De werking van de GENI monitor wordt later in de scriptie uitgebreid besproken.

Hoofdstuk 2

SFA-Architectuur

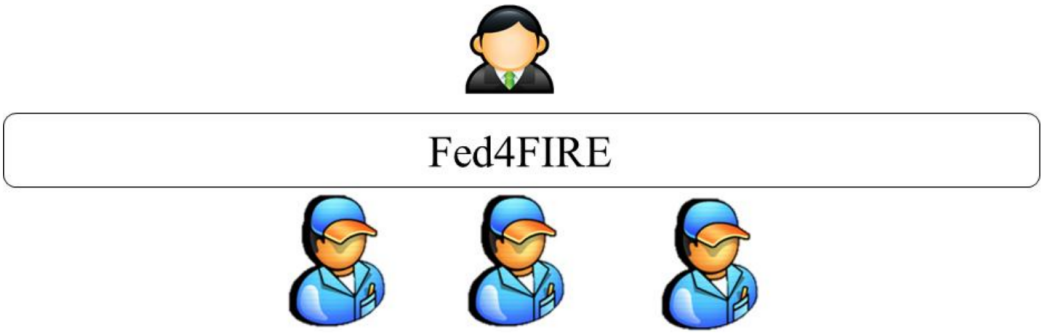
Deze masterproef zal als onderdeel van FIRE, een Europees onderzoeksproject naar een innovatief internet, een monitoringsservice maken met een bijhorende API. FIRE gebruikt voor zijn testbedden de SFA architectuur. Deze architectuur is ontwikkeld om de configuratie en aansturing van testbeds of aggregates over de hele wereld gelijk te maken. Binnen een aggregate hebben onderzoekers een slice waarin ze hun experiment kunnen uitvoeren. Deze slice bestaat uit een aantal componenten, slivers genaamd. Hierdoor moeten onderzoekers maar één configuratie leren, waarmee ze vervolgens op elk testbed kunnen werken.

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, het delen van testbeds makkelijker maken, waar te maken. Doordat alle testbeds op een andere manier werkten, was het voor een onderzoeker erg moeilijk om verschillende testbeds te gebruiken. Een onderzoeker moest eerst kennis maken met de specifieke configuratie van een testbed, alvorens hij ermee kon werken. Aangezien dit zeer tijdrovend is, is het noodzakelijk om de configuratie van testbeds gelijk te trekken.

Een belangrijk begrip is een federation. Dit begrip heeft vooral te maken met authorisatie. Binnen een federatie worden testbeds, service en onderzoekers vertrouwd. De bedoeling van Fed4FIRE is dan ook de testbeds of aggregates binnen FIRE samen te voegen in 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 onderste personen) is de service provider. Fed4FIRE voorziet de link tussen beide partijen.

SFA voorziet een aantal functionaliteiten. De eerste 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 slices aan te maken. Een slice is een container waarin een experiment draait.

Verder moet het mogelijk zijn voor eigenaars om de autorisatie te controleren. Hierbij is het bijvoorbeeld mogelijk om maar een beperkt aantal mensen toegang te verlenen.

2.2 Entiteiten

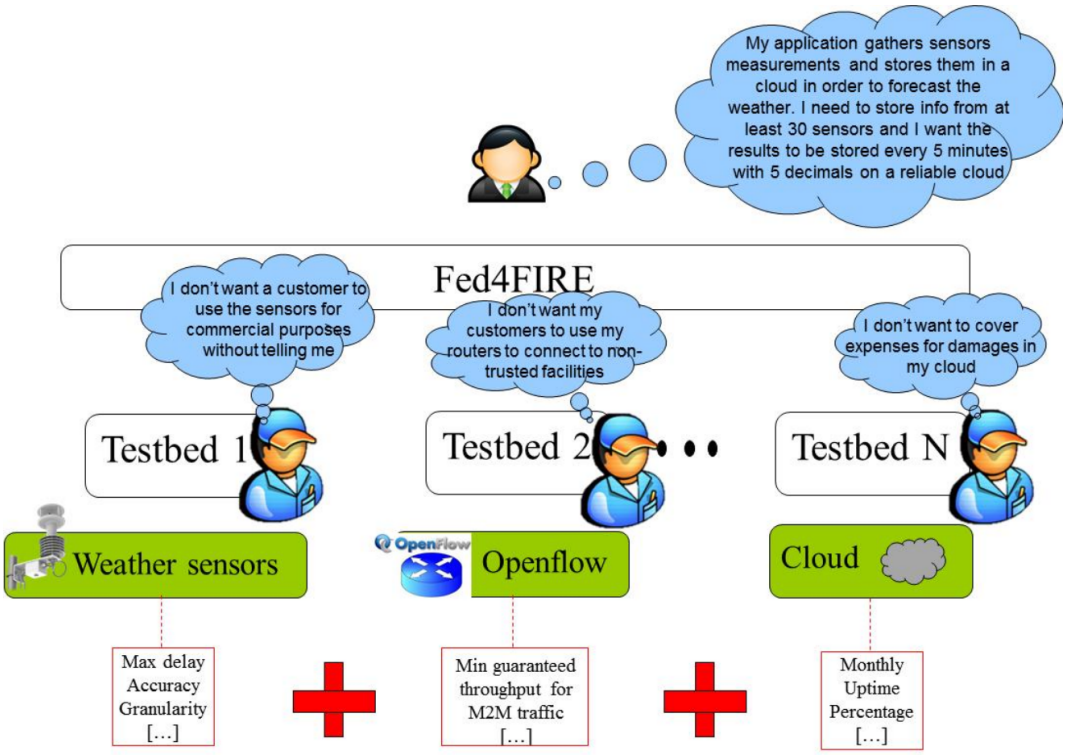
SFA herkent 4 entiteiten.

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

Wat hierop 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). 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 geeft een voorbeeld van een aantal mogelijke beleidsregels.



Figuur 2.2: Eigenaars bepalen het beleid van hun testbed.

2.2.2 Operators

Operators voorzien het onderhoud van het testbed. Dit onderhoud omvat o.a. herstellingswerken, beveiliging, voorkomen van schadelijke activiteiten.

2.2.3 Researchers

De onderzoeker is de klant. Hij gebruikt een testbed om experimenten uit te voeren. Deze experimenten verlopen in 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

Een testbed is opgebouwd uit meerdere onderdelen die opgedeeld kunnen worden in meerdere lagen.

2.3.1 Component

Een testbed bestaat uit vele onderdelen. Een primaire bouwblok van een testbed is een component. Een component is bijvoorbeeld een computer, router of programmeerbaar access point. 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 of aggregaten. Een aggregaat is een verzameling componenten die onder eenzelfde beheerde valt. Zo is de virtual wall2 van iMinds een aggregaat omdat het beheer van dit volledige testbed onder iMinds valt.

2.3.3 Aggregate manager

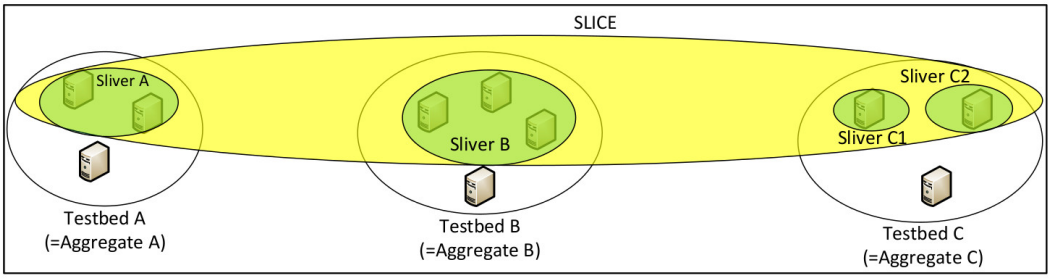
Elke aggregate wordt beheerd door een AM (aggregate manager). De aggregate manager is een stuk software dat een interface aanbied aan onderzoekers. Via deze interface kan bijvoorbeeld een slot gereserveerd worden om een experiment op te zetten. Een aggregate manager vervult taken zoals 'stukken van het testbed', slices genaamd, toe te wijzen aan onderzoekers of een experiment.

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

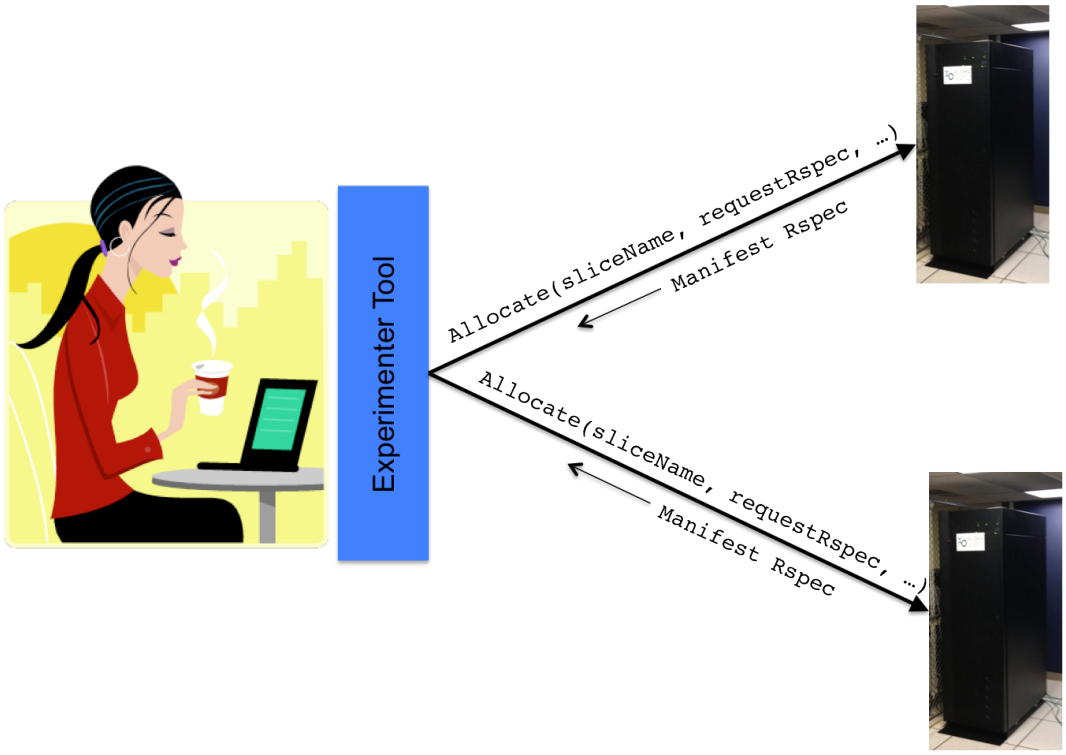
Een slice is een verzameling van slivers. Een slice is een abstract begrip dat omschreven kan worden als een container waarin een experiment draait. Vanuit het perspectief van de onderzoeker komt dit overeen met de testopstelling die hij ter beschikking heeft. Vanuit het perspectief van de owner of eigenaar 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. Een slice kan over meerdere aggregates heen kan lopen.



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

2.4 RSpec

Een RSpec (Resource Specification) , is een xml file die een proefopstelling beschrijft(GENI, 2014e). RSpecs kunnen opgedeeld worden in 3 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.

Voor een onderzoeker is een RSpec een beschrijving van een experiment. 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.

Hoofdstuk 3

Analyse en probleemstelling

Zoals eerder besproken, zal deze masterproef een monitoringsAPI maken 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 (Future Internet Research and Experimentation), een project voor de verbetering van onderzoek naar internet en netwerken, in de gaten houden. Dit hoofdstuk zal de werking van de FIRE monitor beschrijven. De FIRE monitor is de monitoringsservice die bij aanvang van de masterproef de monitoring verzorgde. Daarna zal er dieper ingegaan worden op de probleemstelling. Vervolgens wordt ook de GENI (Global Environment for Network Innovations) monitor kort besproken. GENI is een Amerikaans project met gelijkaardige doelstellingen als FIRE.

3.1 FIRE Monitor

iMinds is het onderzoekscentrum waar de masterproef uitgewerkt wordt en is tevens leidinggevend in het FIRE project. 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).

3.1.1 Componenten

De monitoringsservice bestaat uit verschillende componenten. De eerste component is de Facility monitoring. De monitoring wordt gebruikt bij de FLS (first level support). De first level support heeft als doel om de basiszaken 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 infrastucture monitoring. Deze component is gericht op componenten binnen een experiment. De verzamelde gegevens omvatten o.a. aantal verstuurde pakketten, aantal verloren pakketten, cpu-load, ...

Een 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). Deze heeft het 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-iLab.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 monitoringssite. De laatste 2 kolommen zijn van minder belang. De eerste kolom geeft de naam van het testbed weer. Daarnaast wordt het resultaat van de laatste ping test getoond. De volgend 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.

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 zal kijken of het mogelijk is om een netwerk op te zetten tussen verschillende testbeds. Zie Figuur 3.2

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 zichtbaar in Figuur 3.3, is het ook mogelijk om de geschiedenis van deze testen op te vragen.

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 resultaten

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 test 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 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 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.

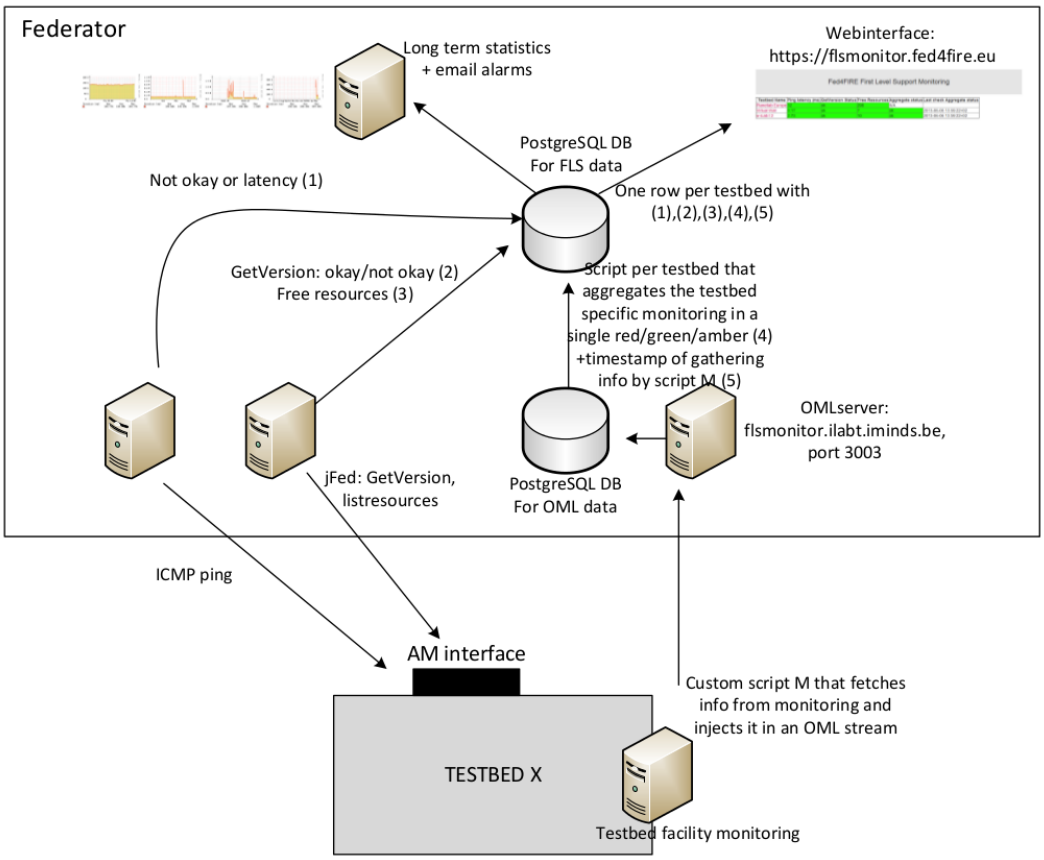
Fed4FIRE First Level Support Monitoring

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-iLab.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. Hierbij wordt gekeken of de latency beneden een waarschuwingswaarde 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 geÃ¯mplementeerd in de masterproef.

Rechtsboven is de webview weergeven, deze haalt de resultaten rechtstreeks uit de databank. Linksboven is de federator weergegeven. De federator is een component binnen een federatie. Deze component mag vrij ingevuld worden en wordt gebruikt binnen een federatie om het beheer vlotter te laten verlopen. Zo worden in dit voorbeeld de statistieken op lange termijn bijgehouden.



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 geÃ¼pdatet. 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 geÃ¼pdatet.

3.2 Probleemstelling

3.2.1 Bereikbaarheid

Het voornaamste probleem is niet de structurering van de gegevens, maar de bereikbaarheid. In de vorige versie zijn de gegevens 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 montoringsAPI. Deze zorgt ervoor dat de gegevens vlot beschikbaar zijn via aanvragen aan de API.

Het ontwerpen van deze monitoringsAPI is het eerste deel van de masterproef. Deze API moet een vlotte toegang tot de resultaten garanderen. Ook moet het mogelijk zijn om deze resultaten de filteren. Een filter die er zeker noodzakelijk is, is het opvragen van de laatste resultaten van een testbed. Eenmaal deze API af is, kan alle communicatie met de achterliggende databank verlopen via de API. 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.

In een tweede deel moet ook gekeken worden naar bestaande monitoringssystemen. Zo heeft GENI (Global Environment for Network Innovations) eveneens een dergelijk framework. Integratie met dit framework zal de samenwerking tussen beide partijen bevorderen. De GENI monitor wordt later in dit hoofdstuk besproken.

3.2.2 Structuur

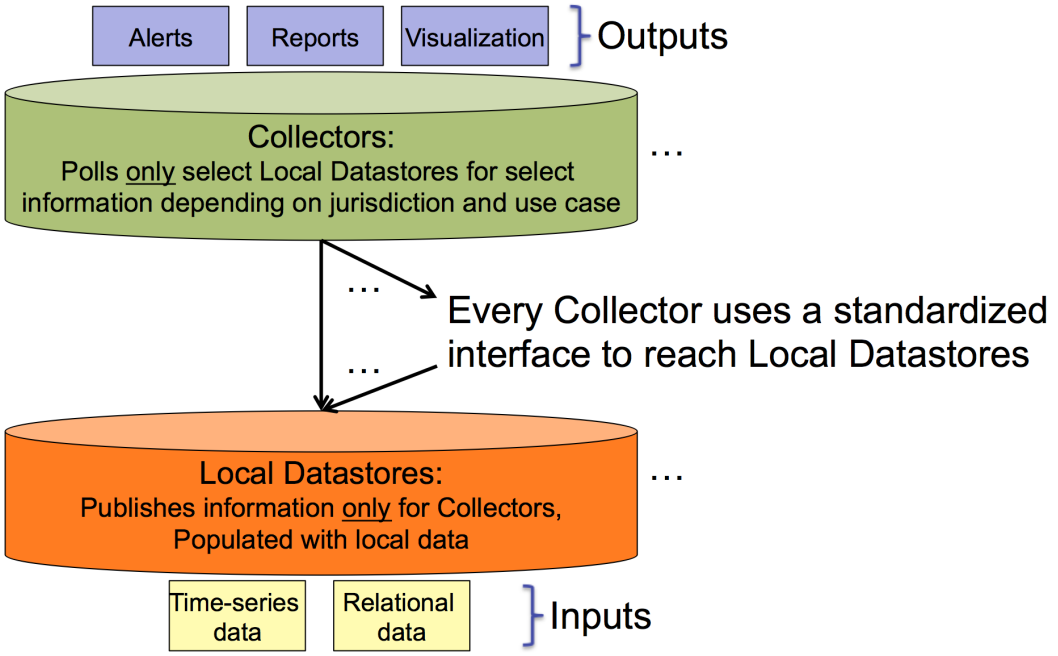
Naast de bereikbaarheid is er ook een structureel probleem. De testen van de FLS monitor hebben een eenvoudige structuur. De testen zijn eenvoudig omdat ze slechts een beperkt aantal parameters nodig hebben en een enkelvoudig resultaat teruggeven. Zo geeft een pingtest de ping waarde terug. Een listResources test geeft het aantal beschikbare resource 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, elke stap heeft een tussenresultaat. Hierdoor volstaat een kolom niet meer. De oplossing die vroeger aangewend is, is het gebruiken van een tweede databank. In die databank is er niet voor elk tussenresultaat een kolom voorzien, maar heeft men de resultaten samen genomen in 2 tussen resultaten. Deze manier lost het probleem van het variabele aantal tussenresultaten op, maar geeft geen proper overzicht van alle tussen statussen. Voor deze informatie moet men zich wenden tot de log.

3.3 GENI monitor

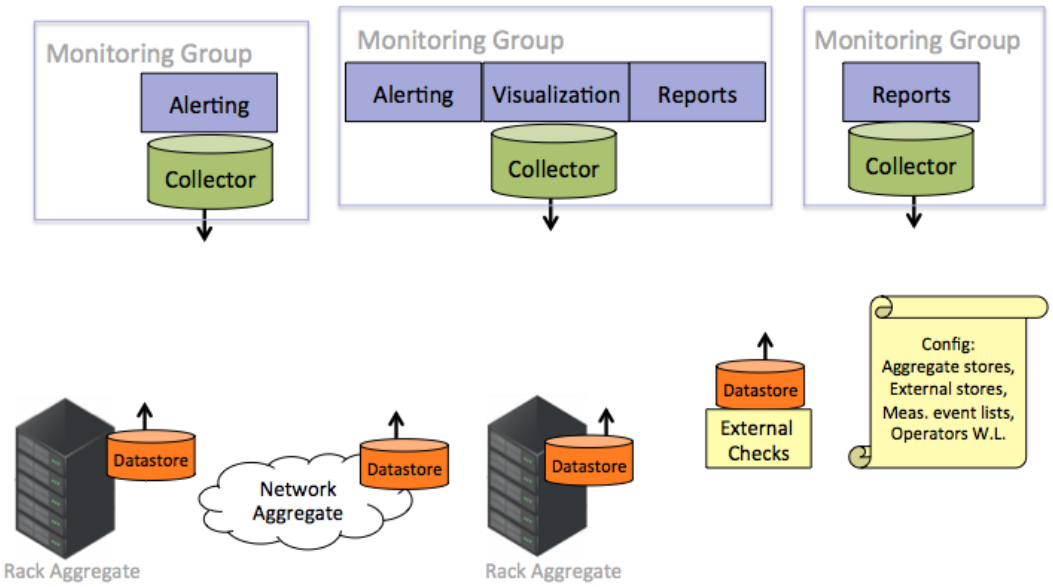
Als er binnen FIRE een nieuw monitoringAPI gemaakt moet worden, is het nuttig om onderzoek te doen naar bestaande oplossingen hiervoor. Zo beschikt GENI over een monitoringsframework (GENI, 2014c), dat hieronder uitgelegd wordt.

GENI (Global Environment for Network Innovations) is een Amerikaans project dat, net zoals FIRE, een virtueel testlaboratorium heeft. Dit lab bestaat uit meerdere onderzoekscentra. Aangezien GENI en FIRE beide bezig zijn met onderzoek naar innovatieve netwerk- en internet ontwikkelingen, 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. Deze datastores hebben een REST polling API om informatie op te halen. Deze API wordt aangesproken door een collector. Een collector verzamelt de gegevens. Dit principe is geschetst in Figuur 3.6



Figuur 3.7: Een collector kan verschillende delen data ophalen.

Een collector kan meerdere testbeds pollen. Hierbij moeten niet alle data opgehaald worden, het is mogelijk dat één collector resultaten van een test op verschillende testbeds ophaald. Een collector zal de nodige gegevens ophalen om de aangesloten monitoringsapplicaties te laten werken. Normaal gezien zal een monitoringsapplicatie een collector gebruiken, al is dat geen vereiste. Dit is beschreven in Figuur 3.7. De boxen zijn de monitoringapplicaties. Deze hebben elk verschillende gegevens nodig, weergegeven in blauw. Elke monitoringsapplicatie heeft een collector (groen). De collector zal de nodige gegevens pollen van een datastore (oranje).

De intense samenwerking tussen FIRE en GENI maakt testbeds beschikbaar voor onderzoekers. 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 of er uitbreidingen nodig zijn aan de datastoreAPI. De GENI datastore API is nog in ontwikkeling.

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 complexe databank. Deze databank houdt zowel de resultaten als de configuratie van de testen bij. Ook is databank voorzien voor het opslaan van tussenresultaten. Als laatste onderdeel is er de integratie met GENI. Hierbij zal onderzocht worden wat er mogelijk is en waarvoor er nog uitbereidingen nodig zijn.

Hoofdstuk 4

Structuur

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 bied 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

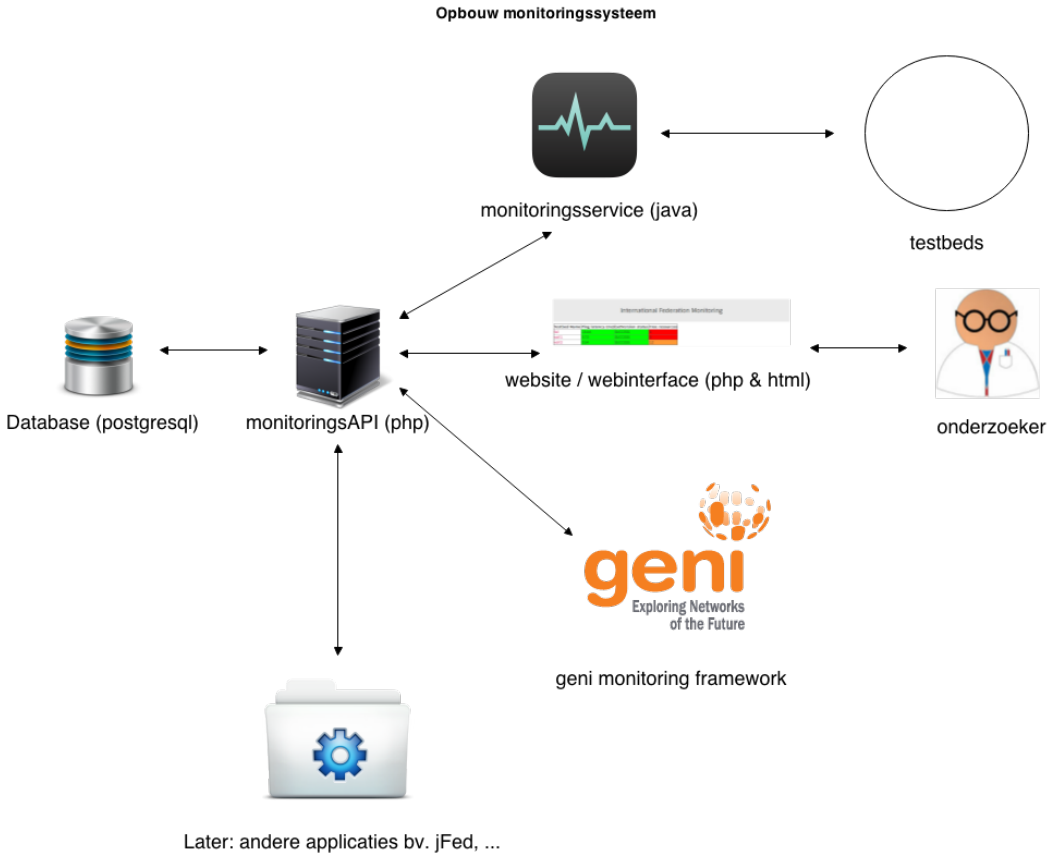
De masterproef bestaat uit een aantal projecten. Volgende sectie bespreekt hun verband en waarvoor elk deel verantwoordelijk is.

1. Een database 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 website om de monitoringsinfo weer te geven.

Daarnaast zijn er nog 2 projecten weergegeven. Dit zijn verbindingen met projecten die niet binnen deze masterproef uitgewerkt worden.

1. Geni monitoringframework, hiervoor wordt de interface van een datastore geïmplementeerd.
2. Later: andere aplicaties 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.

4.1.1 MonitoringsAPI

Dit onderdeel vorm de kern die alle andere projecten aan elkaar bindt. De monitoringsAPI staat in voor de communicatie tussen de buitenwereld met de databank. Enerzijds worden er resultaten toegevoegd aan de databank. Deze resultaten zijn afkomstig van de monitoringsservice. 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.

4.1.2 Database

De database is verantwoordelijk voor het bijhouden van informatie. Dze informatie kan opgedeeld worden:

1. Configuratie van testen:
   (a) De testbeds die door de monitoringsservice gecontroleerd moeten worden.
   (b) De users die gebruikt worden voor authenticatie op de testbeds.
   (c) Welke testen er zijn, hierbij moet het mogelijk zijn om nieuwe testen toe te voegen zonder te veel verandering aan te brengen in de code.
   (d) De scheduling, hierbij moet het mogelijk zijn om elke test met een verschillend interval uit te voeren.
2. Resultaten: 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 zal dan ook verlopen via de monitoringsAPI.

4.1.3 Monitoringsservice

Dit deel zal de testen uitvoeren. 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.

4.1.4 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.

4.1.5 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 zal de monitoringsinformatie die hij nodig heeft ophalen van verschillende datastores. Een collector wordt gebruikt door een applicatie om de nodige informatie op te halen.

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 zaken 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 GENI api.

4.1.6 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 en vervolgens deze gebruiken.

Hoofdstuk 5

Technologieën

Dit hoofdstuk bespreekt de gebruikte technologieën in de masterproef. Samengevat maakt deze masterproef een monitoringsservice met bijhorende API die de monitoringsinformatie beschikbaar moet stellen aan de buitenwereld. De monitoringsinformatie wordt opgeslagen in een PostgreSQL databank. De API werkt met PHP en is gehost met een apache HTTP-server. Voor de website is naast HTML, gebruikt gemaakt van PHP om de API calls af te handelen. Voor opmaak en layout werd ook gebruik gemaakt van CSS en javascript.

5.1 Onderdelen

Zoals te zien op Figuur 4.1 bestaat de masterproef uit een aantal delen.

1. Database die alle data zal bijhouden.
2. Een monitoringsAPI die de kern vormt waarmee alle andere projecten verbonden worden.
3. Een monitoringsservice voor het uitvoeren van de testen zelf.
4. Een website om de monitoringsinfo weer te geven.

Hieronder worden alle gebruikte technologieën besproken. Door de objectgeörienteerde opbouw is het uitwisselen van technologieën mogelijk. Doordat de software moet kunnen draaien op alle soorten platformen, waaronder ook linux platformen, is er echter niet gekozen voor platformafhankelijke talen zoals .net. In plaats daarvan is er gekozen voor platformonafhankelijke talen zoals java, perl , python, php , ...

5.1.1 Databank - PostgreSQL

Figuur 5.1: PostgreSQL logo

De databank die gebruikt wordt is een postgreSQL databank. PostgreSQL is een open-source object-relational database systeem(PostgreSQL, 2014). Met meer dan 15 jaar ervaring heeft het een sterke reputatie voor stabiliteit en betrouwbaarheid opgebouwd. PSQL (PostgreSQL) werkt op alle grote platformen en met alle prominente programmeertalen.

Het gebruik van PSQL werd door het bedrijf in kwestie, iMinds, opgelegd omdat het vorige systeem ook met PSQL werkte. Hierdoor waren de ontwikkelaars al vertrouwd met dit data-systeem. Er zijn een aantal voordelen om PSQL te gebruiken. Een eerste voordeel is terug te vinden in de vele functionaliteiten die in PSQL ingebouwd zijn. Een ander voordeel is de uitgebreide documentatie die te vinden is op de wiki van PSQL (https://wiki.postgresql.org).

5.1.2 MonitoringsAPI

Figuur 5.2: PHP logo

Voor de monitoringsAPI is gekozen voor PHP (PHP: Hypertext Preprocessor). PHP is een algemene open-source scriptingtaal. PHP kan voor alle doeleinden gebruikt worden, maar is vooral gericht op webontwikkelingen(php, 2014). PHP is al enkele jaren een prominente speler op het gebied van webtechnologie. PHP blijkt hier een goede keuze te zijn doordat er zeer veel documentatie en modules beschikbaar zijn.

Figuur 5.3: Apache logo

Om php te hosten, is er nood aan een HTTP server. Deze zal met behulp van de php code de overeenkomstige pagina genereren. De facto standaard is de Apache HTTP-server. Apache HTTP is een open-source HTTP voor voor moderne operating systemen zoals unix en windows(Foundation, 2014). Het doel van apache Http is om een veilige, efficiënte en uitbereidbare server te maken die overweg kan met de HTTP standaard.

5.1.3 Monitoringsservice

Figuur 5.4: Java logo

Voor de monitoringsservice is er gebruikt gemaakt van Java. Java is een zeer gekende en veelgebruikte programmeertaal die platform onafhankelijk is(Oracle, 2014). Java is een gekende en betrouwbare taal die gebruikt kan worden voor alle mogelijke toepassingen te maken. Omdat de jFed automatedtester in java geschreven is, zal de monitoringsservice ook in java gemaakt worden. Vermits de automated tester, de module die de testen effectief uitvoert, in dezelfde taal geschreven is, is de integratie zeer eenvoudig en efficiënt.

5.1.4 Website

Voor de website wordt gebruikt gemaakt van php, deze php code zal eerst een call doen naar de API. Eenmaal het antwoord van de call ontvangen is, zal de html code gegenereerd worden. Voor de layout is gebruik gemaakt van css en javascript.

Hoofdstuk 6

Uitwerking

Deze masterproef maakt een monitoringsservice met een bijhorende monitoringsAPI die de monitoringsinformatie beschikbaar stelt. De API zal de resultaten bijhouden in een databank. Deze databank bevat de configuratie gegevens, de resultaten en de beschrijvingen van de verschillende testen. De API doorloopt voor elke aanvraag een opeenvolging van stappen. Zo wordt eerst de aanvraag geparset, 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.

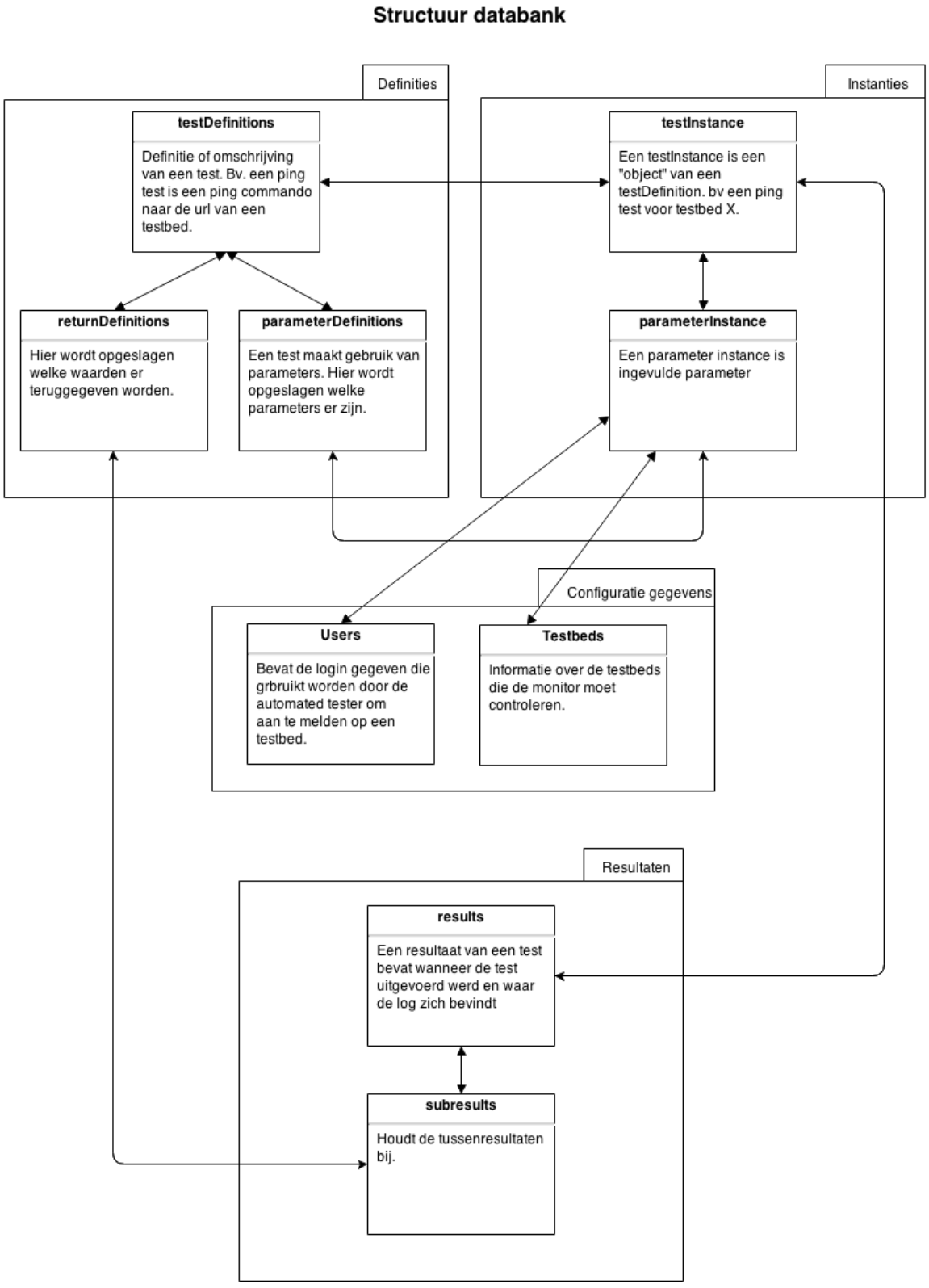
6.1 Databank

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 6.1 op volgende pagina. Hier zijn de tabellen gegroepeerd op basis van functionaliteit om een beter overzicht te behouden.



Figuur 6.1: De stuctuur van de databank

6.1.1 Definities

De eerste groep tabellen bevat de definities. De definities bevatten een omschrijving van een test. Hierbij worden de parameters en 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.

6.1.2 Instanties

Deze tabel houdt de instanties bij. Een testinstantie is de test zelf. Als de vergelijking met object-georiënteerd 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 systeem de nodige flexibiliteit om de nieuwe definities aan te maken. Net zoals bij de definities zijn de ingevulde waarden hier ook ondergebracht in een aparte tabel. Dit is met dezelfde reden, namelijk het toelaten van een variabel aantal parameters.

6.1.3 Configuratie gegevens

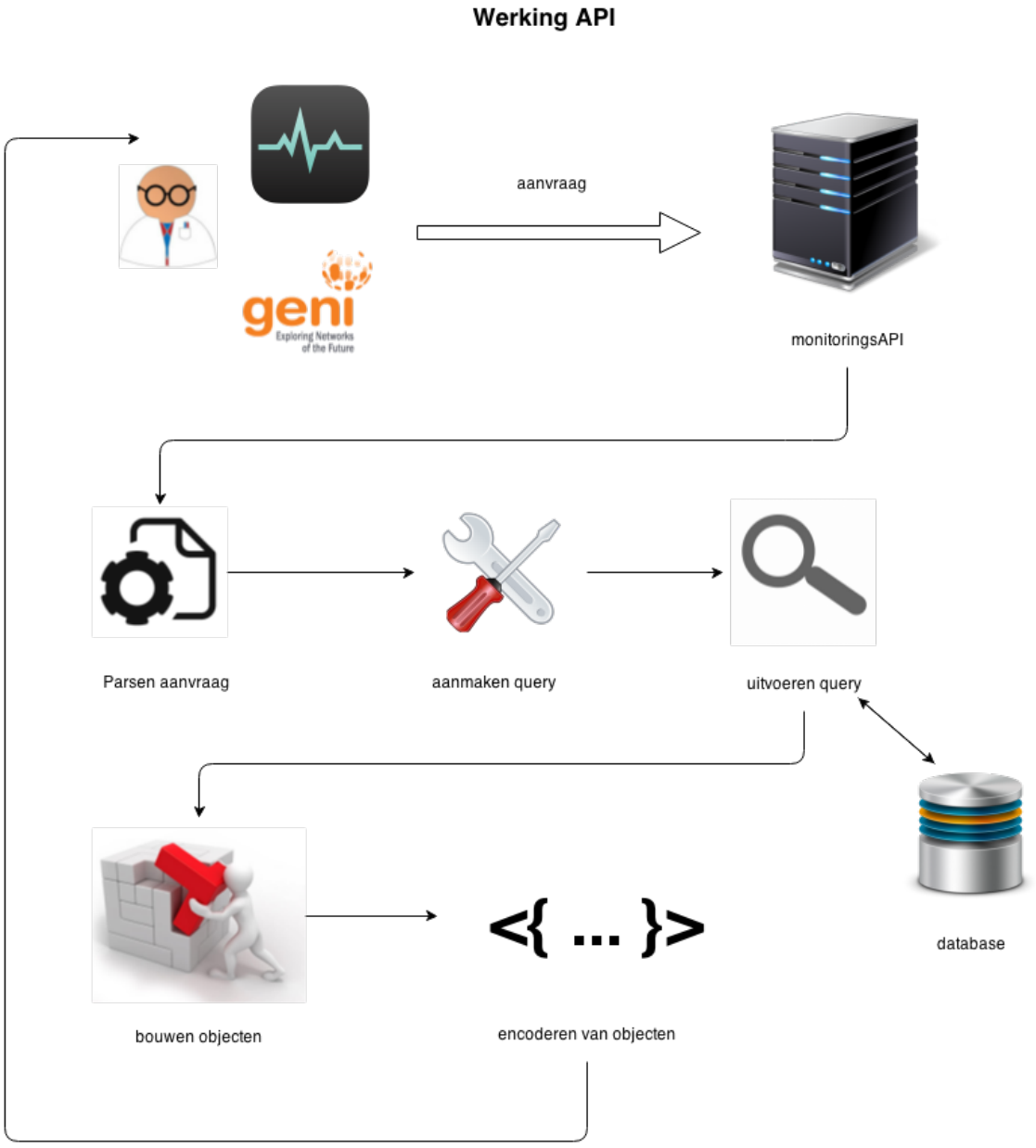
De configuratie gegevens bestaan uit de gebruikers en testbeds. De gebruikers zijn belangrijk omdat ze de login informatie bevatten die gebruikt worden door de automated tester om testen uit te voeren. Zo zal een login test een gebruiker nodig hebben die de juiste rechten heeft op dat testbed. Naast 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.

6.1.4 Resultaten

Deze tabel zal alle resultaten bijhouden. Elk resultaat heeft een bijhorende logfile, deze wordt momenteel niet opgeslagen in de databank, maar apart 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.

6.2 Webservice / API

De webservice zal informatie uit de databank ophalen en omvormen naar objecten. Dit verloopt in een aantal fasen, zoals weergegeven in Figuur 6.2.



Figuur 6.2: De werking van de API

6.2.1 Fasen

De stappen die overlopen worden zijn:

1. parsen aanvraag
2. opbouwen query
3. uitvoeren query
4. bouwen objecten
5. encoderen objecten

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

6.2.2 Parsen aanvraag

Er zijn drie soorten aanvragen, de eerste behandelt het opvragen van configuratiegegevens, terwijl de volgende ophalen en toevoegen van resultaten bevat. De laatste groep aanvragen is noodzakelijk voor de monitoringsservice en betreft het ophalen van testen en aanpassen van scheduling.

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 reference in bijlage B).

Een derde groep aanvragen verzorgt het uitvoeren van de testen en de scheduling:

1. testInstance gefilterd op nextrun: geeft de testen weer die uitgevoerd moeten worden op een zeker tijdsstip.
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.

Voor een complete lijst van aanvragen en filter mogelijkheden wordt verwezen naar de reference die is toegevoegd in bijlage A.

6.2.3 Aanmaken query

In deze stap wordt de query aangemaakt; opgegeven filters worden vertaald naar SQL syntacs. 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 aanvragen te behandelen. Zo wordt de GENI aanvraag door een andere queryBuilder afgehandeld dan een standaard aanvraag.

6.2.4 Uitvoeren query

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

6.2.5 Bouwen objecten

In deze stap worden het resultaat van de query lijn per lijn overlopen en worden objecten gevormd. Door de structuur van de databank worden meerdere lijnen samengenomen om een object te vormen. Dit komt door de aparte tabellen die gelinkt worden, waardoor er voor elke parameter een andere lijn ontstaat.

Om compabiliteit te verhogen worden er geen klassen zoals testbed, user en test aangemaakt, maar wordt alles direct opgeslagen in geneste array[1]. Hierdoor gaat er minder tijd verloren met de aanmaak van objecten die vervolgens terug omgevormd moeten worden naar het specifieke dataschema. Dit wordt gedaan door de fetcher. Opnieuw wordt hier gebruik gemaakt van overerving om andere dataschema's te ondersteunen.

6.2.6 Encoderen van objecten

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

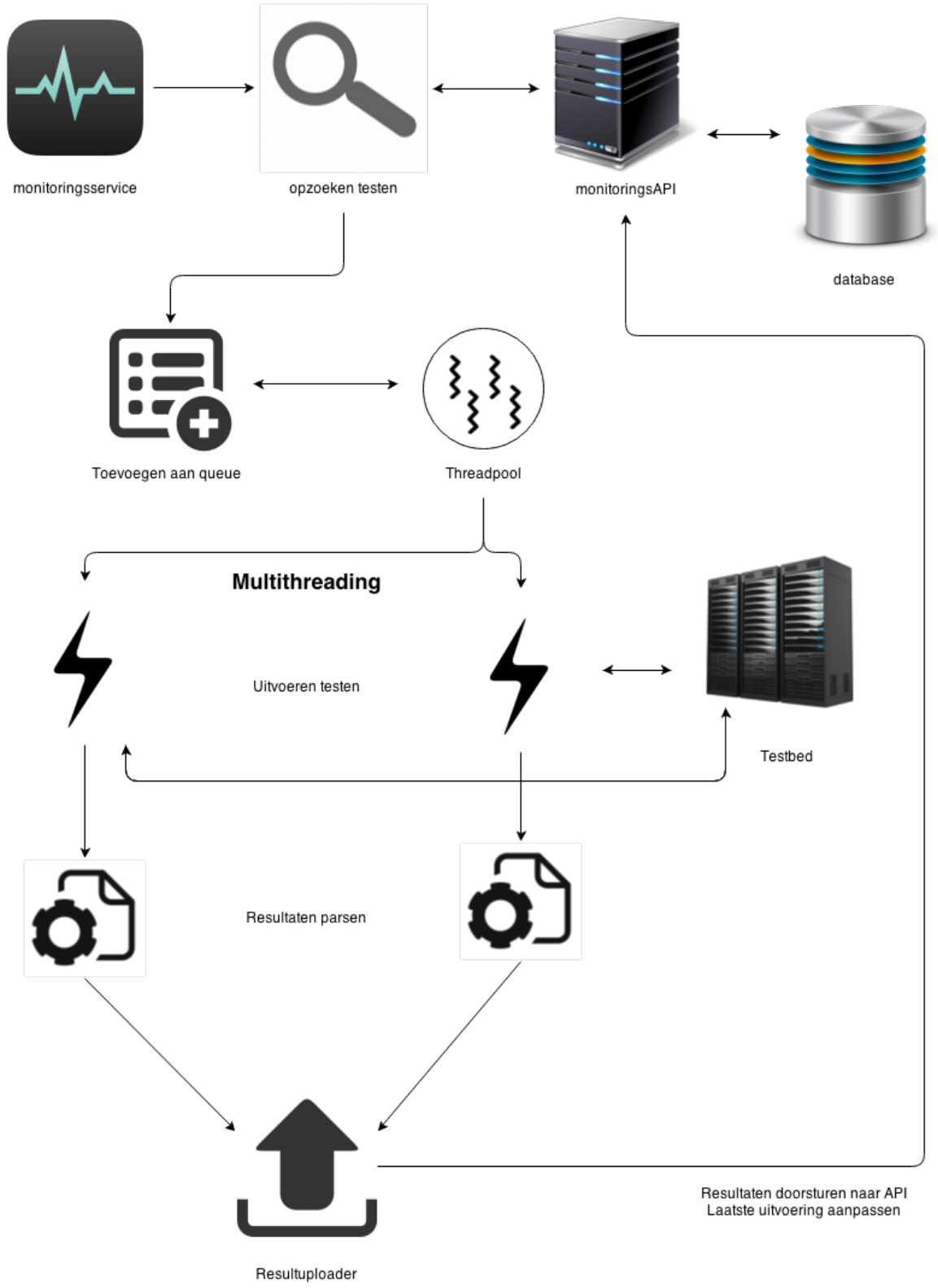
6.3 Service

De service werkt met meerdere threads Er is één hoofdthread die testen ophaalt via de API en toevoegt aan een queue. Vervolgens zal een threadpool de testen een voor een uit de queue halen en uitvoeren. De grootte van de testpool is instelbaar, standaard is dit $cpu_{cores} * 2 + 1$.

Elke test wordt uitgevoerd op een aparte thread. Tijdens de uitvoering wordt verbinding gemaakt met een of meerdere testbeds via de jFed automated tester. Deze geeft een XML-file terug waar de tussenresultaten uit 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 6.3.

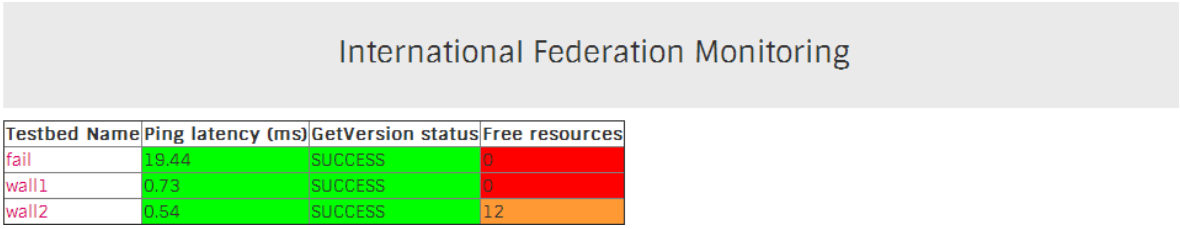
[1] Merk op dat een PHP array zowel een map als een array is.



Figuur 6.3: De werking van de monitoringsservice

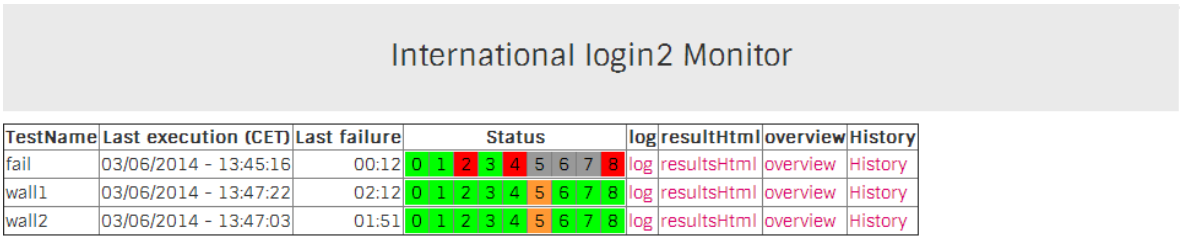
6.4 Website

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 6.4.



Figuur 6.4: FLS monitor

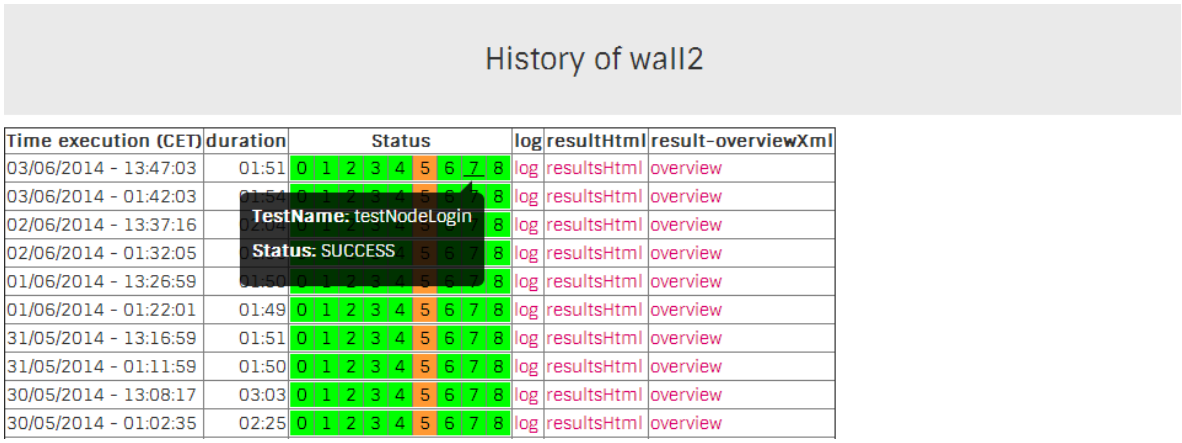
Een volgende interface wordt gebruikt om detail resultaten weer te geven voor één testtype. In Figuur 6.5 wordt een overzicht gegeven van de login testen[2]. Op deze figuur kan eenvoudig afgelezen worden dat de test voor het eerste testbed mislukt is en dat de oorzaak daarvan bij stap twee en vier (in rood) ligt. Diezelfde test is wel gelukt voor wall1 en wall2, maar met een waarschuwing bij stap 5.



Figuur 6.5: Overzicht van resultaten van de login testen.

[2]Login2 in de titel is te verklaren dat dit login testen zijn voor aggregate manager 2

Tenslotte is er nog een overzicht van de geschiedenis van een specifieke test, dit is zichtbaar in Figuurhistnieuw. Wanneer de cursor over een tussenresultaat gaat, verschijnt er een tooltip[3] waarop de naam van de subtest samen met de status wordt weergegeven[4].



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

Figuur 6.6: 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 6.7a en Figuur 6.7b.

[3]Maakt gebruik van powertip, beschikbaar op http://stevenbenner.github.io/jquery-powertip/
[4]Dit is ook mogelijk bij de weergave van een test.

(a) De originele overview in XML formaat

(b) Overview in html formaat

Figuur 6.7: Naast de console uitvoer zijn ook het originele resultaat in XML formaat en het overeenkomstige HTML formaat beschikbaar.

Hoofdstuk 7

Loadtest

Deze masterproef maakt een monitoringsservice met een bijhorende monitoringsAPI. Dit systeem zal testbeds monitoren en deze data toegankelijk maken om de betrouwbaarheid van een testbed te bepalen. Naast betrouwbaarheid is echter ook de performantie van belang. Door een aantal aanpassingen aan de monitoringsservice werd tijdens de uitwerking van de masterproef ook een tool gemaakt die loadtesten kan uitvoeren. Een loadtest is een test waarbij een bestaande testen binnen een kort tijdsinterval meerdere malen uitgevoerd wordt. De loadtest zal eerst een bestaande test ophalen en deze vervolgens op meerdere threads tegelijk starten.

7.1 Doel

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 leerkrachten 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. Een voorbeeld van een uitgevoerde stresstest komt later in dit hoofdstuk aan bod. De bedoeling van deze stresstest is om 50 login testen 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.

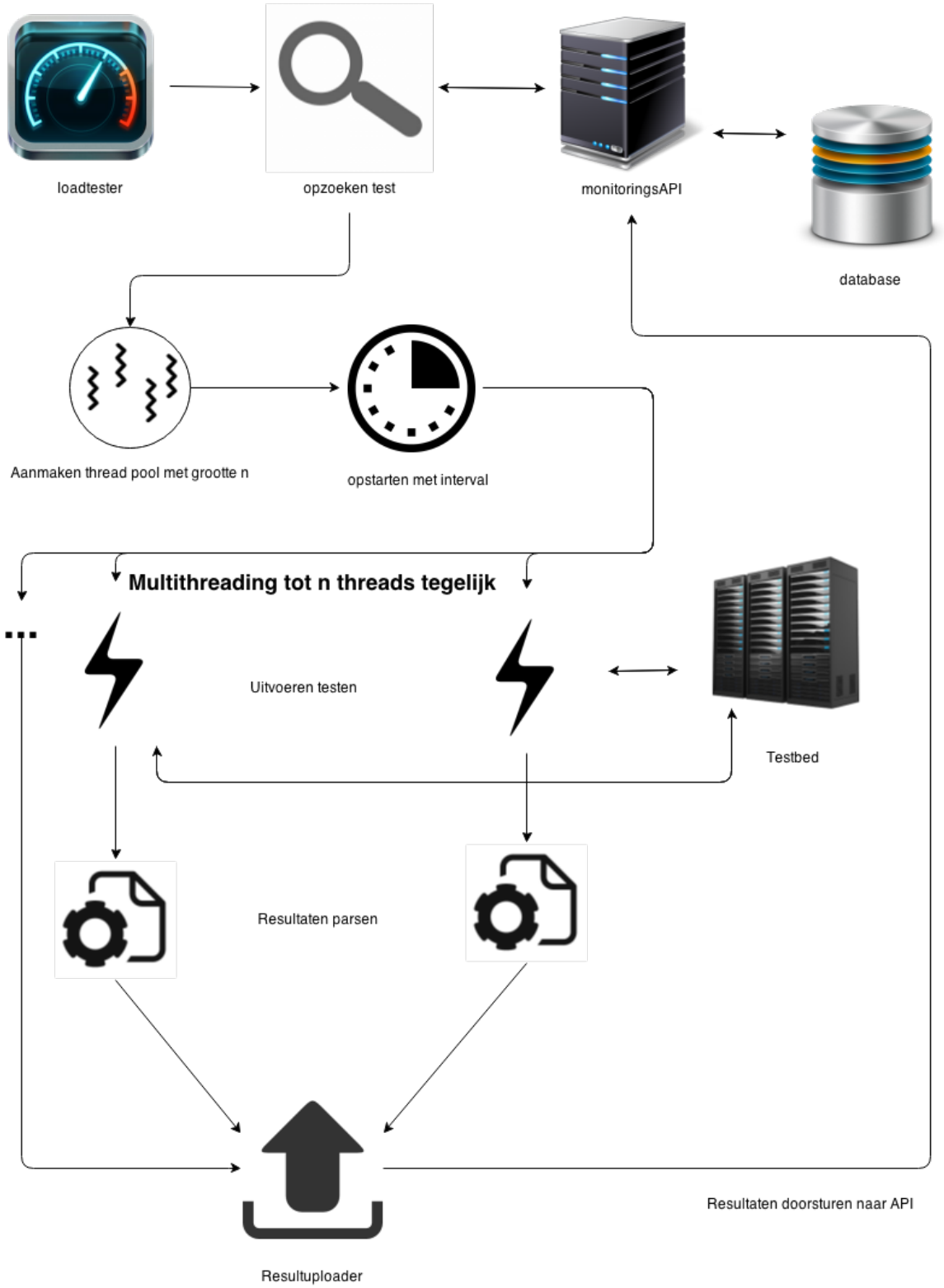
7.2 Uitwerking

De uitwerking wordt weergegeven in Figuur 7.1. Let op de overeenkomsten met Figuur 6.3, 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.

Figuur 7.1 op volgende pagina geeft de uitwerking weer. 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. Dit is niet mogelijk bij de service, vermits de grootte daar beperkt is. 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.

Na het uitvoeren van de testen zijn de resultaten van de stresstest beschikbaar via de monitoringsAPI. Doordat er in de huidige implementatie geen onderscheid wordt gemaakt tussen resultaten van de monitoringsservice en de loadtesten wordt deze databank en API gekloond. De kloon hiervan zal vervolgens dienen om de resultaten van de loadtesten op te slaan. Samenvoegen van de monitoringresultaten met de loadtestresultaten is mogelijk, maar bleek noch een prioriteit, noch noodzakelijk te zijn.



Figuur 7.1: De uitwerking van de loadtesten.

7.3 Voorbeeld

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 7.2 geeft een overzicht van de gebruikte computers. Deze computers zijn per twee opgedeeld in slices. Figuur 7.3 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.

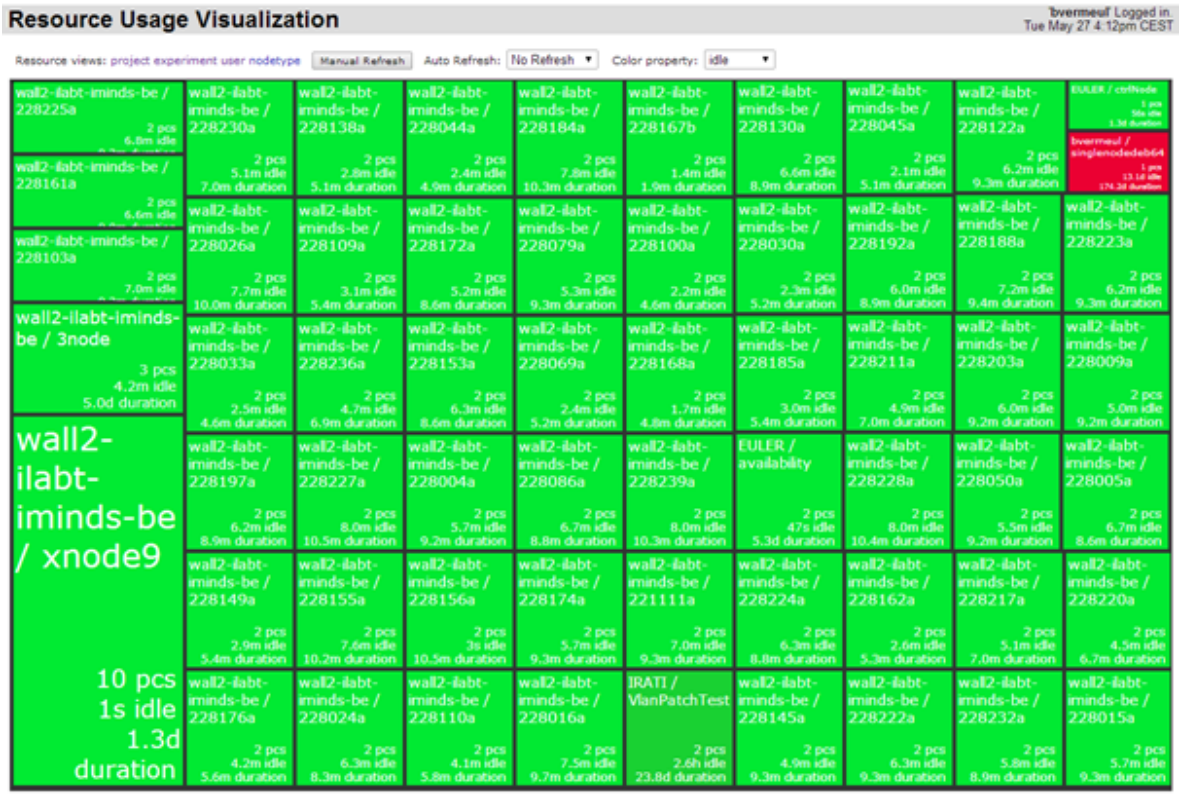
Project wall2-ilabt-iminds-be is using 119 PCs!

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 7.2: Overzicht van de verschillende slices.



Figuur 7.3: Status van de slices.

Vermits er in Figuur 7.3 maar één pc rood is, kan besloten worden dat het testbed dergelijke belasting kan verwerken. Figuur 7.4 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 7.4: 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 7.5. Ook hier moeten de percentages verdubbeld worden wat een belasting van 90% geeft over een langere periode.

Figuur 7.5: 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 monitoringsinterface[1]. Figuur 7.6 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.

[1]Deze 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

Figuur 7.6: 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 IBCN/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.

[1] Of 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 explaination 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 clausules or contact IBCN/iMinds.

Arguments

1. testbed: the testbed of the corresponding result
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 timezone.
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
&testbed=wall2
```

Will return all the results of each[2] pingtest on wall2 between 18/03/2014 18:29:00 and 19/03/2014 18:29:00.

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

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

[2]Although 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. testbed: the testbed of the corresponding result
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 each[3] pingtest of each testbed.

```
last?testdefinitionname=stitch&count=5&testbed=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.

[3] Although 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. testbed: the testbed used by the test
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 >= 19/06/2014 12:00:00; used by the monitoringsservice.

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 monitoringservice 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. aggragate: 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_available[4] test for wall1 and wall2.

[4]The 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 monitoringsAPI 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 recources 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("nodelogin");
        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 plaatsen[1].

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);";
```

[1]Dit 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 aangemaakte testen.

Controleer of alle parameternamen, 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.

A. S. Foundation (2014). Apache http server project. URL http://httpd.apache.org/.

GENI (2014a). About geni. URL https://www.geni.net/?page_id=2.

GENI (2014b). The geni api. URL http://groups.geni.net/geni/wiki/GeniApi.

GENI (2014c). Geni operational monitoring project. URL http://groups.geni.net/geni/wiki/OperationalMonitoring.

GENI (2014d). Overview of operational monitoring. URL http://groups.geni.net/geni/wiki/OperationalMonitoring/Overview.

GENI (2014e). 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.

Oracle (2014). Learn about java technology. URL http://www.java.com/en/about/.

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.

php (2014). Preface. URL http://www.php.net/manual/en/preface.php.

PostgreSQL (2014). About. URL http://www.postgresql.org/about/.

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.