

# Thesis

SDF Monitoring

Thesis information	
Student Name:	Simon Rosman
Student Number:	1519616
First Examiner:	Marten Wensink

## Document Information

General	
Document Title:	Thesis
Area or Project:	SDF Monitoring
Document Owner:	Simon Rosman
File Name:	Thesis.docx
Attachments:	Monitoring Design (Conceptual Solution) Use Case 3 Design

History					
Version	Date	Status	Author	Affected pages:	Approved by:
v0.1	14-Oct-2009	New	Simon Rosman	All	
v0.2	24-Nov-2009	Update	Simon Rosman	All	
v0.3	30-Nov-2009	Update	Simon Rosman	All	
v0.4	03-Dec-2009	Update	Simon Rosman	All	
V1.0	14-Dec-2009	Final	Simon Rosman	All	Wijnand van Plaggenhoef, Marten Wensink

Distribution			
Version	Name	Date	Publication Status
v0.2	Marten Wensink, Wijnand van Plaggenhoef	24-Nov-2009	Draft
v0.3	Wijnand van Plaggenhoef	30-Nov-2009	Draft
v0.4	Wijnand van Plaggenhoef	03-Dec-2009	Draft
v1.0	Wijnand van Plaggenhoef, Marten Wensink, Hogeschool Utrecht	15-Dec-2009	Final

Copyright © 2009 by Cordys Corporation B.V. ("Cordys"). All rights reserved; subject to limited distribution and restricted disclosure only. Cordys Integrator, Cordys Orchestrator, Cordys Studio, and Cordys Portal are trademarks of Cordys Systems B.V. All other trademarks mentioned herein may be/are the trademarks or registered trademarks of their respective owners and should be noted as such. The information in this document is confidential, constitutes the proprietary property of Cordys, and is protected by copyright laws and international copyright treaties. No part of this document may be reproduced or transmitted in any form or by any means, electronic or mechanical, for any purpose, without the express written permission of Cordys. The information contained in this document is subject to change without notice. Cordys does not warrant that the information contained in this document is error free. Cordys assumes no liability for any damages incurred, directly or indirectly, from any errors, omissions, or discrepancies between the software and the information contained in this document.

# Thanks

---

First of all, I would like to thank my uncle Johan Rosman. Without his help, finding a company to do my internship would have taken a lot more time!

I would like to thank Wijnand van Plaggenhoef as well for giving me the opportunity to do this internship in the SDF department and for his guidance throughout the project.

Also thanks to Gert Smits for the countless times he helped out in discussions about the development of a solution. I also want to thank Marten Wensink, my school counselor, for his advice about writing the plan of approach and the thesis, and for his encouragements.

I wish you pleasant reading, and I hope you will learn from it!

Simon Rosman

# Summary

---

This document describes the project about monitoring the Cordys Multi-cluster Environment. This is the Cordys & Google environment.

The purpose of the project is to create a solution which enables Operational Management to overview & monitor the status of the entire Cordys multi-cluster environment.

The document describes the Conceptual Solution which covers a full solution for the projects purpose.

The 'smallest path' of this Conceptual Solution was proved with three Use Cases, in a testing environment.

# TABLE OF CONTENT

<b>Thanks.....</b>	<b>3</b>
<b>Summary.....</b>	<b>4</b>
<b>1. Introduction .....</b>	<b>8</b>
1.1 Content .....	8
1.2 Audience .....	8
1.3 Purpose of the document .....	9
1.4 Definitions, acronyms, and abbreviations.....	9
<b>2. Environment.....</b>	<b>10</b>
2.1 The Cordys Company .....	10
2.2 Project history .....	11
2.3 Related projects.....	11
2.4 Roles & Stakeholders.....	12
2.4.1 Project executor.....	12
2.4.2 Roles.....	12
2.4.3 Stakeholders .....	12
2.5 Technical Environment .....	13
2.5.1 The Cordys Solution .....	13
2.5.2 Cordys Cluster.....	13
2.5.3 Cordys multi-cluster environment .....	15
2.5.4 SaaS Deployment Framework (SDF).....	15
2.6 Organizational environment .....	16
<b>3. Problem, Purpose, Job.....</b>	<b>17</b>
3.1 Problem description .....	17
3.2 Purpose .....	17
3.3 Job.....	17
<b>4. Conceptual Solution .....</b>	<b>18</b>
4.1 Introduction .....	18
4.2 Requirements .....	18
4.2.1 Preparing the interviews .....	18
4.2.2 Conducting the interviews .....	18
4.2.3 Processing the results.....	18
4.2.4 The results; the requirements.....	19
4.2.4.1 Key functional requirements.....	19
4.2.4.2 Key non-functional requirements.....	19
4.3 Definitions .....	19
4.4 Positioning the monitoring tool.....	19
4.4.1 Description.....	19
4.4.1.1 Scenario 1.....	20
4.4.1.2 Scenario 2.....	20
4.4.2 Decision.....	20
4.5 Monitoring Integration .....	21
4.5.1 Description.....	21
4.5.2 Decisions .....	21
4.6 Monitoring individual Cordys components .....	21
<b>5. Implementation.....</b>	<b>22</b>

5.1 Introduction .....	22
5.2 Success Use Cases .....	22
5.2.1 UC1: Web gateway failure.....	23
5.2.2 UC2: SOAP processor failure .....	23
5.2.3 UC3: Engine task in 'waiting' state on Admin Cluster caused by error on Customer Cluster.....	24
5.3 Testing environment .....	25
5.4 The Monitoring tool: Nagios.....	26
5.4.1 Monitoring tool selection .....	26
5.4.2 How Nagios works .....	26
5.4.2.1 Objects.....	26
5.4.2.2 Checks .....	26
5.4.2.3 Configuration.....	26
5.4.2.4 Web interface .....	26
5.5 Distributed Nagios environment .....	27
5.5.1 Nagios Slave .....	28
5.5.1.1 OCSP .....	28
5.5.1.2 OCHP.....	28
5.5.2 NSCA .....	29
5.5.2.1 NSCA client.....	29
5.5.2.2 NSCA server.....	29
5.5.3 External command file.....	29
5.5.4 Nagios Master .....	29
5.6 Managing the Nagios configuration .....	30
5.6.1 NConf, the management tool .....	30
5.6.2 Configuration deployment process .....	31
5.7 Monitoring Cordys with Nagios .....	32
5.7.1 UC1: Web gateway failure.....	32
5.7.1.1 Creating the check command .....	32
5.7.1.2 Assigning the check command to a host .....	33
5.7.1.3 Testing the service check .....	34
5.7.1.4 The results .....	34
5.7.2 UC2: SOAP processor failure .....	35
5.7.2.1 Plug-in language.....	35
5.7.2.2 Connecting to Cordys .....	35
5.7.2.3 Authenticating a SOAP call.....	36
5.7.2.4 Writing the plug-in.....	36
5.7.2.5 Using the plug-in .....	36
5.7.2.6 Testing the service checks.....	37
5.7.2.7 The results .....	37
5.7.3 UC3: Engine task in 'waiting' state on Admin Cluster caused by error on Customer Cluster.....	38
5.7.3.1 Problem description .....	38
5.7.3.2 The designed solution .....	38
5.7.3.3 The solution procedure.....	39
5.7.3.4 Writing the plug-ins .....	40
5.7.3.5 Using the plug-ins.....	42

---

5.7.3.6 Testing the service checks.....	43
5.7.3.7 The results .....	45
<b>6. Conclusions .....</b>	<b>46</b>
6.1 The results.....	46
6.2 The process .....	46
<b>7. Recommendations.....</b>	<b>47</b>
7.1 BPM vs. Provisioning Ticket comparison .....	47
7.1.1 Defining matching conditions .....	47
7.1.2 Thresholds .....	47
7.1.3 Provisioning models.....	47
7.2 Securing the communication between Nagios instances.....	47
7.2.1 Users and Passwords.....	47
7.2.2 Configuration deployment.....	48
7.2.3 Check results .....	48
7.2.4 Ticket lists / Process Instance lists.....	48
7.3 Nagios implementation options.....	49
7.3.1 Nagios slave-master communication failure .....	49

# 1. Introduction

---

In a company such as Cordys, a large and complex system is used to offer services to customers. In order to keep customers satisfied, the amount of problems must be kept to a minimum. And if any problem might occur, solving it as quickly as possible is most important.

This phenomenon is called Operational Management. The project described in this thesis is about assisting Operation Management in their job of guarding the systems status, and assisting in case problems do occur. The people responsible for Operational management have indicated they would like to have a solution which reduces the necessity for manually checking the status of the entire system.

This is called monitoring, and therefore, this project is titled SDF monitoring. SDF stands for Software as a Service Deployment Framework, and can be regarded as the system. This will be explained in more detail later.

## 1.1 Content

### **Chapter 1, Introduction**

Is the current chapter.

### **Chapter 2, Environment**

Introduces the Cordys Company. Also, it described the trigger to this project, along with the history of this project and the related projects.

Next, it describes the technical environment, which is necessary in order to fully understand the technical context.

### **Chapter 3, Problem, Purpose, Job**

Describes the problems with the current environment, or the 'Business Case' of the project. Additionally, it describes the goals of the projects and the job that must be done.

### **Chapter 4, Conceptual Solution**

Introduces the proposed Conceptual Solution, which has been designed during this project. Parts of this conceptual solution are: Functional and Non-functional requirements for a monitoring setup.

### **Chapter 5, Implementation**

Introduces the success conditions that must be met. It introduces the testing environment, the selected monitoring tool, the setup of the monitoring tool, management of the monitoring tool. Most importantly, it covers the created solution for proving the Use Cases.

### **Chapter 6, Conclusions**

Evaluates the entire project, and puts a conclusion to it.

### **Chapter 7, Recommendations**

Explains what remaining issues should get attention after this project is finished.

## 1.2 Audience

The intended audiences for this document are:

- SaaS Deployment Framework team
- The student's school
- Operational management (Cordys IT)



## 1.3 Purpose of the document

This document serves a number of purposes, namely:

- To describe what this project was about (the subject).
- To describe how the solution was developed.
- To describe what decisions were made, and why.

## 1.4 Definitions, acronyms, and abbreviations

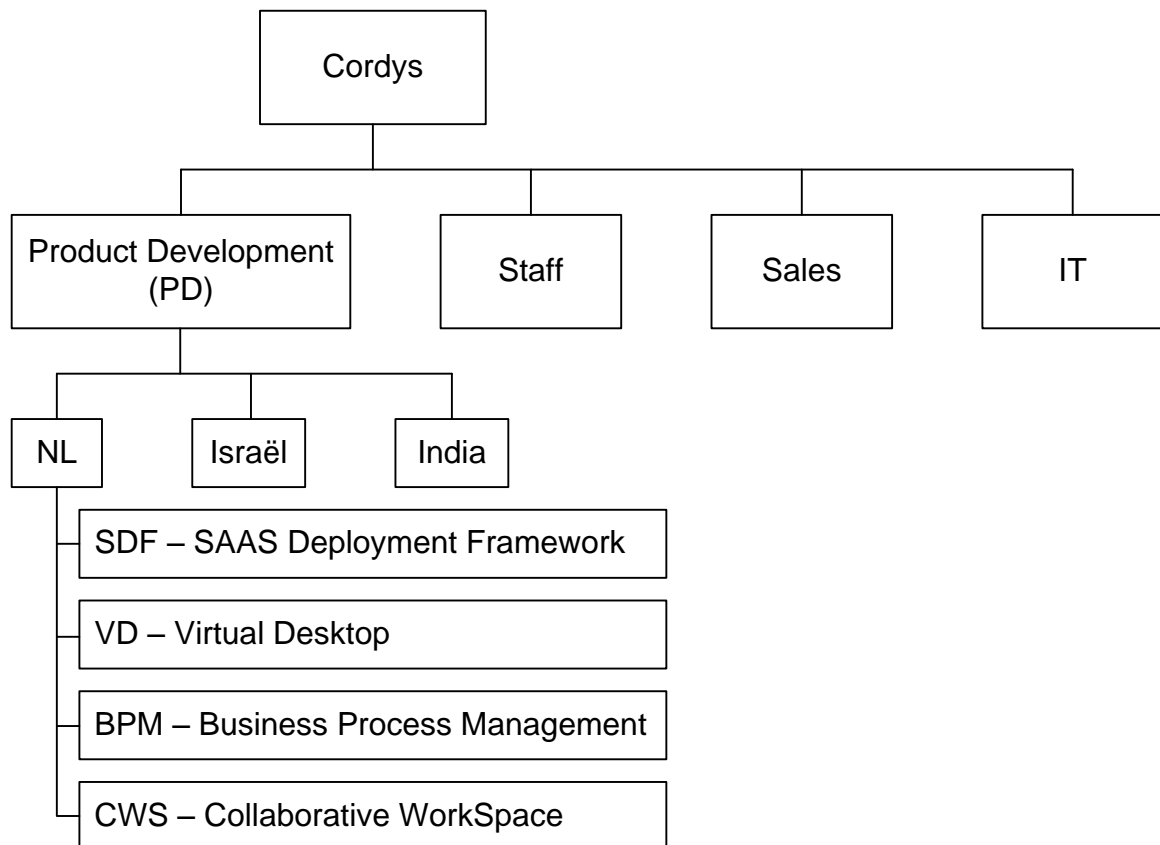
Term	Description
SaaS	Software as a Service
Cordys BPMS	Business Process Management Suite
Cordys Cluster	A group of tightly coupled servers with Cordys BPMS installed.
Provisioning	The process of remotely distributing and managing applications, organizations, users and all related data.
Metering	The process of logging the usage of applications and services.
SDF	SaaS Deployment Framework The Cordys product used for provisioning and metering across a number of clusters.
JMX	Java Management Extensions JMX enables a Java application to be remotely monitored.
JMX counter	This document uses this term as follows: A JMX component, placed in an application, which allows external applications to retrieve information, such as status, from a Java application.
OSI Model	Open System Interconnection Reference Model A description of computer communication in 7 layers. Layer 7 is abstract, layer 1 is the simplest.
UC	Use Case Description of the behavior of a system in a certain situation in relationship to its users (a.k.a. actors).
SOAP	Simple Object Access Protocol
SAML	Security Assertion Markup Language
DOM	The Document Object Model is a platform- and language-neutral interface that will allow programs and scripts to dynamically access and update the content, structure and style of documents.
XML	Extensible Markup Language XML is a simple, very flexible text format used for data exchange.
PIM	Process Instance Manager
PI / BPM	Process Instance / Business Process Model A Process Instance is a Business Process Model in execution.

## 2. Environment

### 2.1 The Cordys Company

Cordys B.V., founded in 1998, is an international company which focuses on software development. Cordys has branches in the Netherlands, the U.S.A., India and Israel. Cordys Business Process Management Suite (BPMS) is the main product Cordys is developing. It consists of several components which offer companies a complete solution to manage and automate business processes. All components were designed and built customer-separated, allowing multiple customers per environment.

The Cordys organization has the following departments:



*Figure 2.1.1 Cordys organizational overview*

The Product Development department is the largest department and consists of approximately 300 employees (approx. 50 in the Netherlands, approx. 10 in Israel and approx. 240 in India). Its job is to maintain, expand and manage the development of the Business Process Management Suite. Each sub-department develops a different BPMS component.

The IT department maintains and manages all computer systems, networking devices and other peripherals. They also manage connections between the company branches and datacenters.

This project is run within the SDF (SAAS Deployment Framework) team. This project directly affects the SDF.

## 2.2 Project history

In the past, a similar project about monitoring has been done. However, the project had a different focus. The project was about monitoring Cordys installations from customers. Customers with their own Clusters wanted to monitor their Cordys installations with their existing monitoring tool(s).

This project differs from the previous project because multiple Cordys Clusters have to be monitored. It is also different because this is about monitoring Cordys' own situation.

## 2.3 Related projects

This project is related to several other projects.

- The project described in chapter 2.2 (Project history)
- The Cordys Google project  
Cordys has started a project with Google to open an application store and various online business solution products. The monitoring solution is created keeping in mind the Cordys + Google environment requirements.

## 2.4 Roles & Stakeholders

### 2.4.1 Project executer

The student has this role. This person is responsible for executing the project, communicating to all stakeholders, project progress and all related tasks.

### 2.4.2 Roles

#### **End user**

This person is an end user of the project results, i.e. he or she will use the created solution.

#### **Project Manager**

This person owns the project and has the highest authority in making decisions.

#### **Product Architect**

This person guides and advises in designing and building the project solution.

#### **School Counselor**

This person is responsible for helping, controlling, checking and evaluating the student during the internship.

### 2.4.3 Stakeholders

#### **SDF Team**

- End user(s):  
All team members
- Project Manager:  
Wijnand van Plaggenhoef
- Product Architect:  
Gert Smits

#### **IT Team**

- End user(s):  
Janamanchi Venkatesh; Chalasani Anil Kumar

#### **School**

- School Counselor:  
Marten Wensink

## 2.5 Technical Environment

Basic knowledge of the Cordys environment is necessary to understand the project. Therefore this document will give a brief explanation of the Cordys environment.

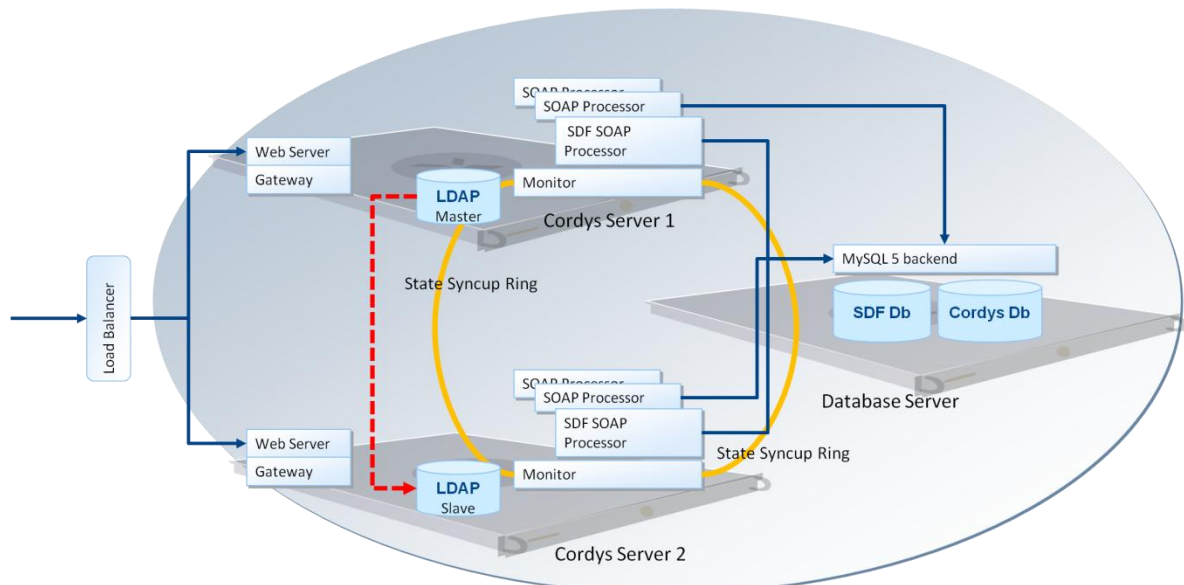
### 2.5.1 The Cordys Solution

Cordys delivers a single platform which allows organizations to design, execute, monitor, change and continuously optimize their critical business processes and operations. This is supported by Cordys BPMS, a set of software. As with all software, it needs an IT infrastructure to operate. Cordys uses the concept of clusters to run the BPMS software.

### 2.5.2 Cordys Cluster

Cordys uses a standardized environment for running the Cordys BPMS. This setup is called a cluster. A Cordys Cluster is a single installation of Cordys BPMS across a number of machines or blades linked into a single State SyncUp ring. A token is passed around. Each machine passes the token to the next machine in the ring. If a machine fails, the ring is rebuilt, and the other machines continue functioning normally.

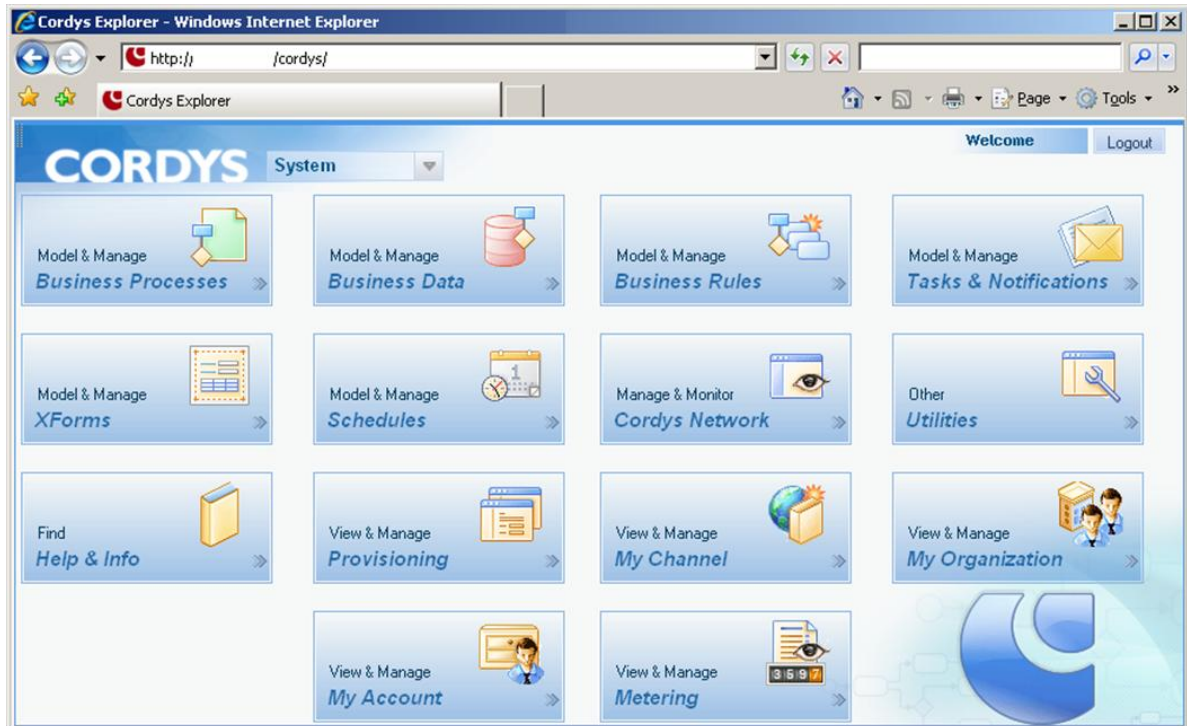
This technology takes care that services on all machines are connected in a way that enables load balancing and failover.



*Figure 2.5.1: An example of a Cordys cluster with 2 Cordys machines in a State SyncUp ring, and a dedicated database server.*

All services within the cluster can be managed from a single user interface, transparent of where the services are physically located.

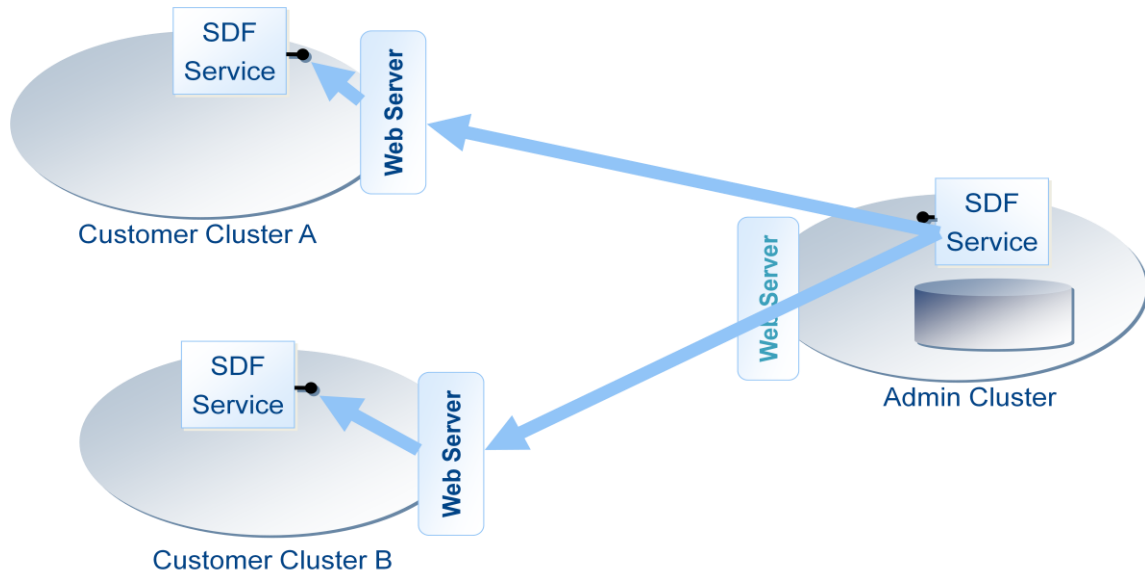
This is shown in the following screenshot, which shows the Cordys interface from where any Cordys component can be managed.



*Figure 2.5.2 The Cordys interface focuses on managing a complete cluster, regardless where a component is physically located.*

### 2.5.3 Cordys multi-cluster environment

Cordys has a number of clusters, and is deploying more clusters. This is the environment which is used for the Cordys Google project. See chapter 2.3, (Related projects), for more details. Currently, there is a cluster intended for Customers. This is called a Customer Cluster. Cordys has another cluster, at a different location, which is built for managing Customer Clusters. This is called (the) Admin Cluster. This combination of multiple clusters is called the Cordys multi-cluster environment. This is shown in the following figure.



*Figure 2.5.3 The Cordys multi-cluster environment*

How are these clusters linked together? This is where the SaaS Deployment Framework shows up.

### 2.5.4 SaaS Deployment Framework (SDF)

SDF is the Cordys component which provides provisioning across clusters. SDF provides the capabilities to create resources such as organizations, users, application subscriptions at the different clusters using provisioning processes (all these terms will be explained in following sections). Different users of the Cordys platform have different requirements about what should be done when a resource is created.

All provisioning processes are based on provisioning models and can be easily customized. Next to the capability of creating resources, SDF also keeps track of those resources and provides the capability to modify or delete those resources, also using provisioning processes.

Lastly, SDF also provides the capability to do metering on the allocation of resources and the use of resources.

SDF supports provisioning of the following items:

#### Organization

Organization provisioning will determine the cluster on which the organization will be created and will then create the organization on the chosen cluster. The SDF bookkeeping on the Admin cluster is updated with the new organization.

#### Application

A customer can request the use of applications and the application provisioning is the process of making the application available within the organization of the customer and also giving the user access to that application. Part of the application provisioning is the process of deploying and configuring the application on the given cluster. This is in SDF 3.0 a

manual process. The SDF bookkeeping on the Admin cluster is updated with the information that the application is available to the organization.

### **User**

User provisioning is the process of creating a user in a given organization. User provisioning is done within the context of an organization and is started with a user registration request (self service) or when an administrator creates the user. In the provisioning process, the cluster of the organization is determined and the authenticated user and organization user are created. The SDF bookkeeping on the Admin cluster is updated with the information that the user is added to the organization. The authenticated user ID is in all clusters and the SDF bookkeeping equal.

## **2.6 Organizational environment**

The SDF team is responsible for updating and maintaining the SDF software. The IT department is responsible for keeping the Cordys multi-cluster infrastructure operational. Together they are the key players in SDF operational management. The results of this project must conform to their requirements. This project is run within the SDF team because SDF is the key player in maintaining and using the Cordys multi-cluster environment. The SDF team is the internal supplier of the SDF software to the IT department. The IT department is responsible for operational management.



## 3. Problem, Purpose, Job

---

### 3.1 Problem description

Whenever a problem arises in the provisioning process, or in the Cordys multi-cluster infrastructure, finding the source of the problem usually takes a long time. The Cordys BPMS does monitor its processes, but it does not monitor the supporting components, such as server hardware, network components and network connections. For example, when a web server is disabled, it results in malfunctioning provisioning processes, but the cause is not clearly visible. In addition, the administrator does not know that something is wrong. The administrator must check manually to ensure all systems are functioning properly. Administrators do not have an overview of the entire Cordys multi-cluster environment.

Compared to the OSI model, only layer 7 (Application) is being monitored. The Cordys BPMS keeps log of events and errors, but logs are not stored at a central location, and they are in various different formats; XML, Plain text and in databases. Some of the log files are too large to be understood correctly.

#### **In short**

Too little information is available of the hardware, network and the infrastructure, and too much, or too distinct information of the SDF to manage the entire environment, and to be able to solve problems adequately.

This problem affects the operational management, and it will make operation management very inefficient and thus expensive to operate.

### 3.2 Purpose

The purpose of the project is to create a solution which enables Operational Management to overview & monitor the status of the entire Cordys multi-cluster environment. The solution must enable administrators to investigate and act quickly in case of problems. The first steps of the solution must be created within 4 months.

### 3.3 Job

The job is to provide a solution which corresponds to the projects purpose.

This solution must:

- Provide a single point from which administrators can overview the Cordys multi-cluster environment, to be referred to as 'Error cockpit'.
- Provide administrators with up-to-date information about the Cordys multi-cluster environment, i.e. monitoring.

As part of this solution the following tasks must be done:

- Listing requirements and creating a design for a new monitoring environment, to be referred to as (the) monitoring design.
- Monitoring a default installation of a Cordys environment, including software and hardware, using an independent monitoring tool
- Expanding SDF with extra JMX counters if necessary, or monitoring existing JMX counters
- Integrate the external monitoring tool and the Error Cockpit.
- Checking and improving the SDF structure regarding monitoring.

## 4. Conceptual Solution

### 4.1 Introduction

The first part of the project consisted of creating a design for the solution which fills the gaps between the requirements from Operation Management and the real environment. This design describes a complete solution for the entire environment. This solution was not implemented completely because it is too big for one project, but the design must provide a conceptual solution which describes the ideal solution for the environment.

See the attached Conceptual Solution for more information.

### 4.2 Requirements

Most important of the solution is that it meets the requirements of the end users. Therefore creating the design was started with listing the requirements of the users. There are several end users in the SDF team, and a few in the IT team. These users were interviewed to retrieve the requirements.

#### 4.2.1 Preparing the interviews

In order to get some useful results from the interviews, a list of questions was created. The first interview consisted of the following questions:

- Which problems do you encounter often?
- Which information sources do you use to determine the source of a problem?
- Which information is missing?
- Which tools are useful when solving a problem?

All questions are about problems, and problem-solving.

#### 4.2.2 Conducting the interviews

The user in India was asked by email about his requirements. Two of the SDF team members were individually interviewed in a conversation.

After conducting the first team member, it turned out that the questions were a little bit incomplete. Most information shows up, but general requirements don't show up specifically.

Therefore, the questions were adapted for the next interview. The following points were added:

- Must have functionality
- Nice to have functionality

Using these points, not only general requirements are retrieved, but also the level of importance is retrieved.

#### 4.2.3 Processing the results

Data from the interview was transformed into Functional requirements and Nonfunctional requirements. This was done by:

1. Merging duplicate requirements. Each user names requirements differently.
2. Grouping related requirements.
3. Choosing a representative name or topic for each group.
4. Rewriting the requirements in Use Case format (User X must be able to do Y).

#### 4.2.4 The results; the requirements

These are the key requirements of the Conceptual Solution which were derived from the interviews.

For a complete list of the requirements, see chapter 5 & 6 of the attached Conceptual Solution.

##### 4.2.4.1 Key functional requirements

###### **Overview of the entire environment**

Administrators must be able to analyze the entire Cordys Multi-Cluster environment in a single overview. This includes admin cluster and customer clusters.

###### **Pushing alerts in case of problem**

Administrators must get alerts in case of any problem.

- This reduces the response time
- This reduces the necessity for manual checking.

##### 4.2.4.2 Key non-functional requirements

###### **Extensibility of Architecture**

It is not possible to foresee the needs of future customers, which customer specific objects need to be monitored and what is needed to do so. For that reason the architecture needs to provide an extension mechanism to add new sets of objects to be monitored.

###### **Independency of architecture**

The monitoring tool must be completely independent of the Cordys product. This means that Cordys must be able to operate while the monitoring tool is not functioning properly, and the monitoring tool must be able to operate without Cordys running.

### 4.3 Definitions

After conducting the interviews, it turned out that almost each different person has a different view on monitoring, and defines monitoring differently. Therefore the conceptual solution must contain the definition of monitoring. The conceptual solution describes the general concept of monitoring, a typical workflow for error handling, and it describes the expected functionality of a monitoring tool.

### 4.4 Positioning the monitoring tool

The requirements describe that monitoring must happen using an independent tool. Therefore the Conceptual Solution starts with describing different scenarios for positioning the monitoring tool in the Cordys environment. This is also important because information must be provided from the entire environment and this information must be accessible by the monitoring tool.

#### 4.4.1 Description

Various architectures of monitoring an environment are possible. Since one of the non-functional requirements was that the monitoring solution must monitor independent of Cordys. This means that monitoring must not be done using Cordys itself. In the past, another project built a solution for monitoring Cordys from within itself. That scenario was considered insufficient and in this design, an external monitoring tool is used.

Two scenarios were created.

#### 4.4.1.1 Scenario 1

In the first scenario, the monitoring tool is positioned completely outside the Cordys environment. In this way, one monitoring instance will be used to check the health of all Cordys systems, for example using the internet.

##### Pros

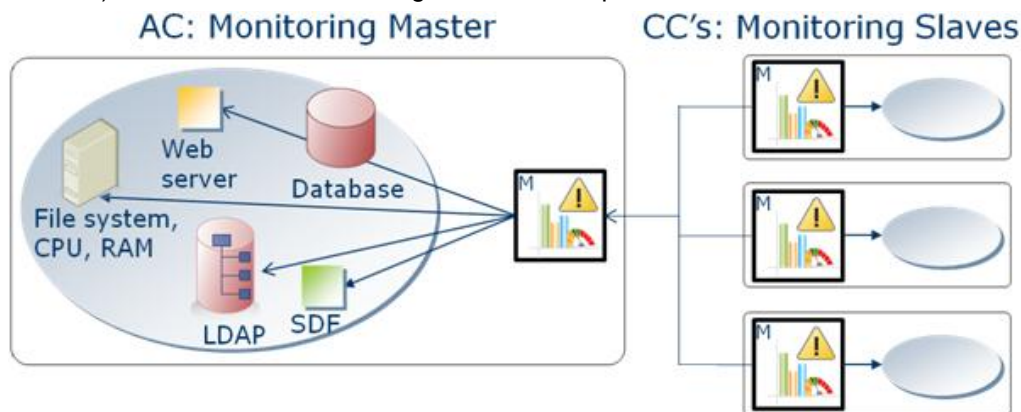
- This reduces the number of monitoring instances (failover is necessary).

##### Cons

- All monitoring data uses the internet connection, so there will be a huge data increase.
- This solution creates security issues because all cluster components must be directly accessible by the external monitoring tool.
- No monitoring in case of connectivity problems.

#### 4.4.1.2 Scenario 2

In the other scenario, each Cordys Cluster gets its own monitoring tool instance. All monitoring instances report to a central monitor, from where administrators can overview the entire system. Since each Cordys Cluster practically has its own network site (physical location), it means one monitoring tool instance per location.



*Figure 4.4.1 Each Cordys cluster has its own monitoring tool instance.*

##### Pros

- This will allow the tool, to detect low level errors.
- All internal cluster components are directly accessible, without accessing them externally.
- In case of internet connectivity problems, each cluster is still being monitored independently.

##### Cons

- The Cordys cluster is exposed to vulnerabilities (bugs) in the monitoring tool.
- This risk can be eliminated by limiting external access to the tool.
- Each Cordys cluster requires its own instance of the monitoring tool.

#### 4.4.2 Decision

Scenario two was chosen because it has the least security risks, and it best matches the Cordys Multi-cluster architecture. Cordys cluster components won't have to be exposed to a monitoring tool outside the Cordys network. Additionally, monitoring load is divided among the clusters.

## 4.5 Monitoring Integration

### 4.5.1 Description

An aspect of the project is, to create a single point of view for administrators. Therefore it is desirable to be able to access all information of the Cordys multi-cluster environment from a single tool.

Since administrators are already using Cordys tools to investigate and analyze problems, it would make sense to be able to use the information from the external monitoring tool from within Cordys.

This will require some sort of integration between Cordys and the monitoring tool, to exchange actual status information.

Three integration scenarios were designed, each using a different level of integration. In the first scenario, no integration is done at all. The monitoring user interface will only display data from the monitoring tool. In the second scenario, a custom user interface must be build into the monitoring tool, allowing it to display data from Cordys directly. In the last scenario, a user interface is built using Cordys technology which allows using existing Cordys tools. The user interface of the monitoring tool is available as a fallback. These scenarios will not be further explained because they were not implemented.

See chapter 7.3 of the attached Conceptual Solution for a complete description of these scenarios.

### 4.5.2 Decisions

A decision, in general, is postponed till the moment, it is required to decide. Using this principle you are not restricted by earlier decisions, which could have been made later.

No decision was made about a scenario for integrating Cordys and the monitoring tool because it was not implemented.

However, while implementing the monitoring tool, the integration aspect was kept in mind. This was done by choosing a monitoring tool which is extensible, and by using open source software.

## 4.6 Monitoring individual Cordys components

Next, proposals for basic procedures to monitor the individual Cordys components mentioned in the requirements were created.

Examples of these procedures are: monitoring a web server, monitoring a database server and monitoring log files.

The conceptual solution doesn't focus on specifying a certain method for monitoring a Cordys component, but this will assist when the design will be implemented, and it gives some understanding of how they work.

This part of the Conceptual solution doesn't describe procedures for monitoring multi-cluster related issues, but for individual Cordys clusters and is therefore the least important part of the Conceptual Solution.

See chapter 8 of the attached Conceptual Solution for the complete list of the procedures and for more detailed information about the procedures.

# 5. Implementation

## 5.1 Introduction

After creating a conceptual solution for the Cordys environment, it was time to put the design to the test. A dedicated testing environment was used to implement the conceptual solution. First of all, it was decided what part of the design should be implemented. This was because the conceptual solution covers a large area of problems, and not enough time would be available to create a solution which completely covers the conceptual solution.

First, the requirements will be described in the form of Use Cases.

Next, since a monitoring tool is required to solve the Use Cases, the following chapters will focus on the monitoring tool; how it works, how it is set up, and how it is managed.

After that, the solution to the Use Cases will be described.

## 5.2 Success Use Cases

Three Use Cases (UC) were written to verify whether the testing environment does its job successfully. These Use Cases describe a certain situation or error which the monitoring tool must be able to detect.

These Use Cases describe the core of the project. The Use Cases were written in order of feasibility.

UC1 basically describes monitoring a generic component, thereby proving that underlying components of Cordys can be monitored.

UC2 moves on a little and describes monitoring a Cordys component using standard SOAP, thereby proving that Cordys internal components can be monitored externally. If a SOAP-processor can be monitored using SOAP, virtually any other Cordys specific component can be monitored.

UC3 takes monitoring to the next level as it describes monitoring across multiple clusters. The Use Case describes monitoring a provisioning task, which is started on the Admin Cluster, and is partially executed on the Customer Cluster. If the Customer Cluster is not able to communicate the results of the provisioning task back to the Admin Cluster, the Admin Cluster will stay in a waiting state. It is currently impossible to detect such situations, and in UC3 the monitoring tool must detect and report such situations.

In all Use Cases, an error is created on purpose, in order to simulate a real problem.

How these errors are created is described in paragraph 5.7.1.3 about Use Case 1, paragraph 5.7.2.6 about Use Case 2 and paragraph 5.7.3.6 about Use Case 3

### 5.2.1 UC1: Web gateway failure

**Short description**

If the web gateway is down, this must be detected by the monitor.

**Scope**

Local cluster

**Purpose**

This Use Case can prove if a monitoring tool is able to provide information about basic cluster components.

**Actors**

- Platform Operator

**Scenario**

1. The monitor checks periodically if the web gateway is functioning properly.
2. Create failure on the web gateway.
3. The monitor detects the error in the web gateway.
4. The monitor informs the Platform Operator.

**Success Conditions**

This Use Case is successful if the Platform Operator gets notified within 2 minutes after the failure.

### 5.2.2 UC2: SOAP processor failure

**Short description**

If a SOAP processor fails, this must be detected by the monitor.

**Scope**

Local cluster

**Purpose**

This Use Case proves if the monitoring tool is able to retrieve information about Cordys specific components using SOAP calls.

**Actors**

- Platform Operator

**Scenario**

1. The monitor checks periodically if the SOAP processors are functioning properly.
2. Create failure on a SOAP processor.
3. The monitor detects the error in the SOAP processor.
4. The monitor informs the Platform Operator.

**Success Conditions**

This Use Case is successful if the Platform Operator gets notified within 2 minutes after the failure.

### 5.2.3 UC3: Engine task in 'waiting' state on Admin Cluster caused by error on Customer Cluster

**Short description**

If a provisioning task which is running on a Customer Cluster fails, this must be detected by the monitor.

**Scope**

Multi-cluster

**Purpose**

This Use Case can prove if the monitoring tool is able to provide previously missing information about the multi-cluster environment. E.g. it tests the complete environment.

**Actors**

- User / Global Operator
- Platform Operator

**Scenario**

1. A Global Operator starts a provisioning task by creating a user account + assigning an application to this user.
2. The task is sent to the customer cluster.
3. The customer cluster starts executing the provisioning task.
4. Create failure on customer cluster which is currently undetectable.
5. The monitor detects the error on customer cluster (e.g. by watching the Process Instance Manager).
6. The monitor updates the Engine task on the Admin Cluster with the error information.
7. The monitor informs the Platform Operator.

**Success Conditions**

This Use Case is successful if the Platform Operator gets notified within 2 minutes after the failure.



### 5.3 Testing environment

The testing environment consists of 3 virtual computers. The setup is shown in the following figure.

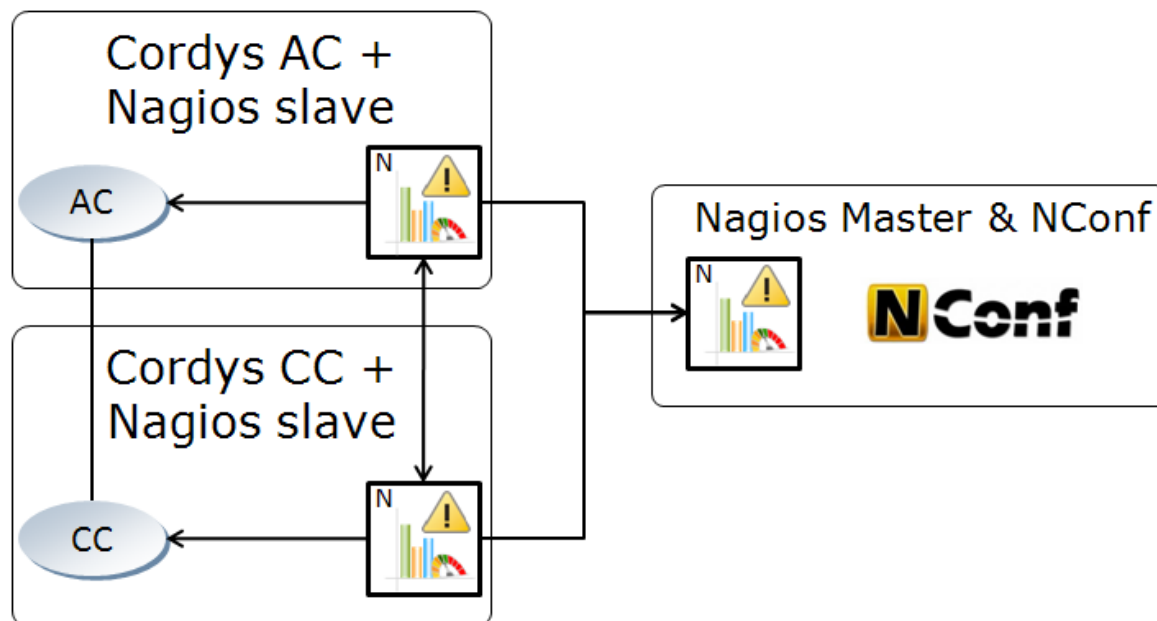






Figure 5.3.1 The diagram of the testing environment

Symbol	Description
	Virtual pc running CentOS Linux.
	Cordys C3 instance AC = Admin Cluster CC = Customer Cluster
	Nagios 3.2.0 instance
	NConf 1.2.5 instance

The two machines on the left represent the Cordys multi-cluster environment where the upper machine is the Admin cluster, and the lower machine is the Customer cluster.

The machine on the right is the Nagios master server. All data from the Nagios slaves is collected here. This is also the machine that stores the Nagios configuration for the entire environment. This is done with the NConf tool.

## 5.4 The Monitoring tool: Nagios

### 5.4.1 Monitoring tool selection

As a monitoring tool, Nagios was chosen for use in this project. This is because Cordys already has experience using Nagios. Customers of Cordys are also using Nagios as a monitoring tool. The tool must be open source. The tool must provide support for: monitoring Web Servers, Distributed monitoring, Notification, Extensibility, SOAP communication, Integration with other systems. Nagios supports this and is therefore used.

### 5.4.2 How Nagios works

To understand the following chapters, it is necessary to understand the basics of Nagios. Nagios uses its own terms for different monitoring processes. The most important will be explained briefly.

#### 5.4.2.1 Objects

Nagios distinguishes two types of objects which can be monitored: Hosts and Services

- A host is a node in a network, for example a router or a server.
- A service is a component on a host, for example a web server, mail server, a log file or CPU load.

Note: A service is always linked to a certain host.

#### 5.4.2.2 Checks

Monitoring an object is done using a check. For hosts, this is a **host check**, for services, this is a **service check**. A check requires a check plug-in which does the actual monitoring by checking the status of a host or a service and returning the answer to Nagios.

Nagios is command based, meaning any command which is executable from a command shell, is usable in Nagios. Because of this, it is very easy to create your own check plug-ins to monitor specific components. This will be further explained in chapter 5.7 where the created plug-ins are described.

Note: A check plug-in must output certain information in order to be useable for Nagios. See [http://nagios.sourceforge.net/docs/3\\_0/pluginapi.html](http://nagios.sourceforge.net/docs/3_0/pluginapi.html) for detailed information on this subject.

#### 5.4.2.3 Configuration

As with most Linux software, Nagios stores its configuration in files. Nagios doesn't provide tools to manage these configuration files. This creates some difficulties in managing multiple Nagios instances in a distributed setup. This will be explained in chapter 5.6 (Managing the Nagios configuration).

Note: For more information about configuring Nagios, and about Nagios itself, see the documentation on [http://nagios.sourceforge.net/docs/3\\_0/toc.html](http://nagios.sourceforge.net/docs/3_0/toc.html)

#### 5.4.2.4 Web interface

Nagios provides a web interface which allows administrators to view detailed status information of all components of the monitored environment. This interface must be installed separately and requires a web server.

## 5.5 Distributed Nagios environment

The Nagios Distributed monitoring is quite complex, and therefore, it will be explained. As explained in chapter 4.4 (Positioning the monitoring tool), one master server is used, and one or more slaves. The following figure shows the different components which are used in the distributed Nagios environment.

### Nagios structure

Nagios Master

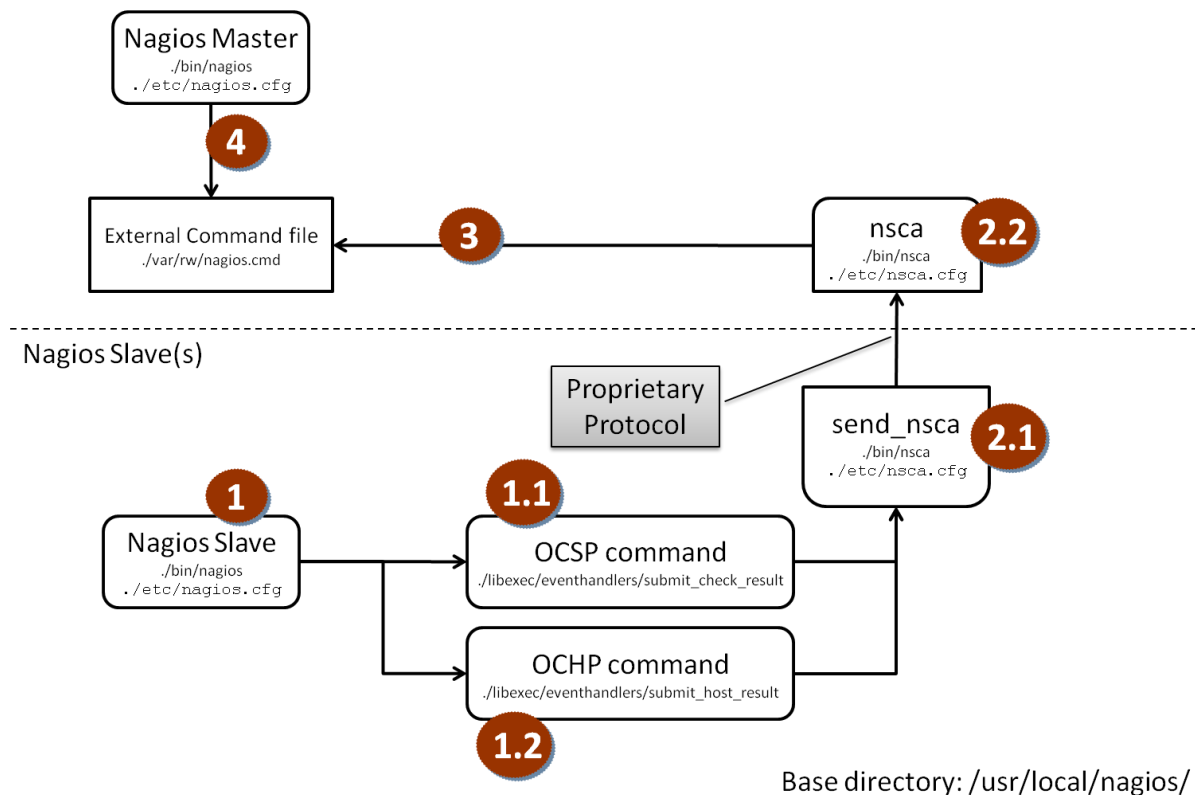


Figure 5.5.1 Nagios distributed setup

The components in the figure are explained in the following paragraphs. The components are described in order of execution. Each time a service or a host is monitored, the following Nagios components are used.

### 5.5.1 Nagios Slave

Monitoring a service or a host starts at the Nagios slave. The Nagios slave executes a check plug-in, which checks the status of a certain Cordys component, and returns the result to Nagios. After receiving the result Nagios executes the command defined in the OCSP or OCHP configuration option. These commands are required for a Nagios instance to send the results of each check to the Nagios master.

#### 5.5.1.1 OCSP

The Obsessive Compulsive Service Processor (OCSP) command is executed by Nagios every time a service check is finished. A script is defined as the OCSP command. Nagios passes the results of the service check as command-line arguments to this script. The script must perform a small translation. The status of the service must be translated from text to a numerical code since the NSCA client only accepts a status code. The script sends the translated service-check results to the NSCA client, which will send them to the Nagios master.

About service status translation:

The following shows the service status descriptions and their corresponding code.

- OK gets translated to 0
- WARNING gets translated to 1
- CRITICAL gets translated to 2
- UNKNOWN gets translated to 3

These service status descriptions and codes are Nagios specific.

#### 5.5.1.2 OCHP

The Obsessive Compulsive Host Processor (OCHP) command is executed by Nagios every time a host check is finished. A script is defined as the OCHP command. Nagios passes the results of the host check as command-line arguments to this script. The script must perform a small translation. The status of the host must be translated from text to a numerical code since the NSCA client only accepts a status code. The script sends the translated host-check results to the NSCA client, which will send them to the Nagios master.

About host status translation:

The following shows the host status descriptions and their corresponding code.

- UP gets translated to 0
- DOWN gets translated to 1
- UNREACHABLE gets translated to 2
- UNKNOWN gets translated to 3

These host status descriptions and codes are Nagios specific.

### 5.5.2 NSCA

NSCA, which stands for Nagios Service Check Adaptor, is an add-on to Nagios which enables Nagios to be used in a distributed setup. NSCA takes care of communicating between Nagios slaves and masters. Any check result from a Nagios slave is sent to the Nagios master by NSCA. On the Nagios slave, NSCA is in combination with the OSCP and OSHP commands. NSCA consists of two parts, a client and a server. It is possible to use authentication and/or encryption between the NSCA client and server, but this was not used in the testing environment because securing inter-cluster communication was not in the scope of the project.

Notes:

- Every check result is sent to the NSCA server individually.
- NSCA does not cache check results or combine multiple check results.
- If sending a check result fails, the result is only stored in the Nagios slave logs. Failed results will not be resent to the Nagios master.

#### 5.5.2.1 NSCA client

The NSCA client is used on the Nagios slaves. The client transfers the results of a service or host check to the NSCA server on the Nagios master. The actual results of the check are supplied as command-line arguments.

#### 5.5.2.2 NSCA server

The NSCA server is located at the Nagios master server and receives service and host check results from the Nagios slaves. After receiving a check result, it writes the result to the Nagios external command file, from which it will be processed by the Nagios master.

### 5.5.3 External command file

Nagios uses an external command file to receive instructions and check results from external programs and plug-ins. Nagios periodically checks the external command file for new commands and check results. After NSCA has written a check result to the external command file, Nagios reads the check result and processes the information inside it.

### 5.5.4 Nagios Master

The Nagios master is responsible for storing, archiving and displaying the service check results. After Nagios has read the check result from the external command file, Nagios updates the host or service status with the received information. If the status has changed, e.g. a service is Critical; Nagios issues an alert and sends out notifications (if it is configured to do so).

## 5.6 Managing the Nagios configuration

An important part of implementing the design is to configure Nagios the correct way. This is especially complicated if Nagios is used in a distributed setup. Nagios itself provides no method to create and maintain Nagios configurations. Therefore an external utility, NConf, is used.

This tool enables administrators to create configurations for a distributed Nagios setup. Other tools are available, but most of them don't support distributed setups.

Even though NConf is designed to create configurations for Nagios, unfortunately it doesn't provide a method to deploy the created configurations to Nagios. This is required in order for NConf to be useful.

The NConf documentation does provide a few suggestions for configuration deployment in its documentation. NConf also provides a sample script for configuration deployment to a single machine.

The next paragraphs describe what changes were made to each component in order to enable administrators to deploy configurations to multiple Nagios instances.

### 5.6.1 NConf, the management tool

In the testing environment, the Nagios master server is also the server on which NConf is running. The following figure shows a screenshot of NConf, after it has generated the configurations.

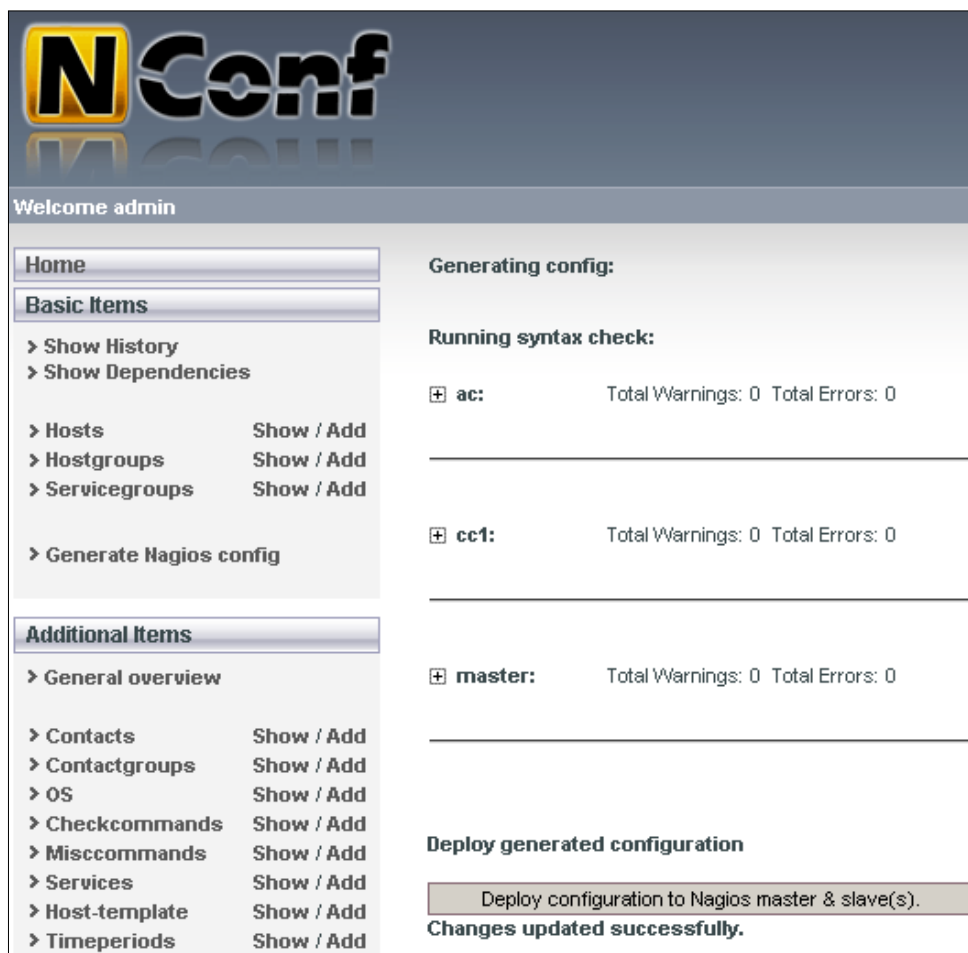


Figure 5.6.1 The NConf configuration screen

If an administrator clicks the configuration deployment button, the deployment page is opened and the deployment process is started.

### 5.6.2 Configuration deployment process

The following figure shows the steps involved in the configuration deployment process. It shows the involved components including their corresponding script files or directories.

#### Configuration deployment

Nagios Master + NConf

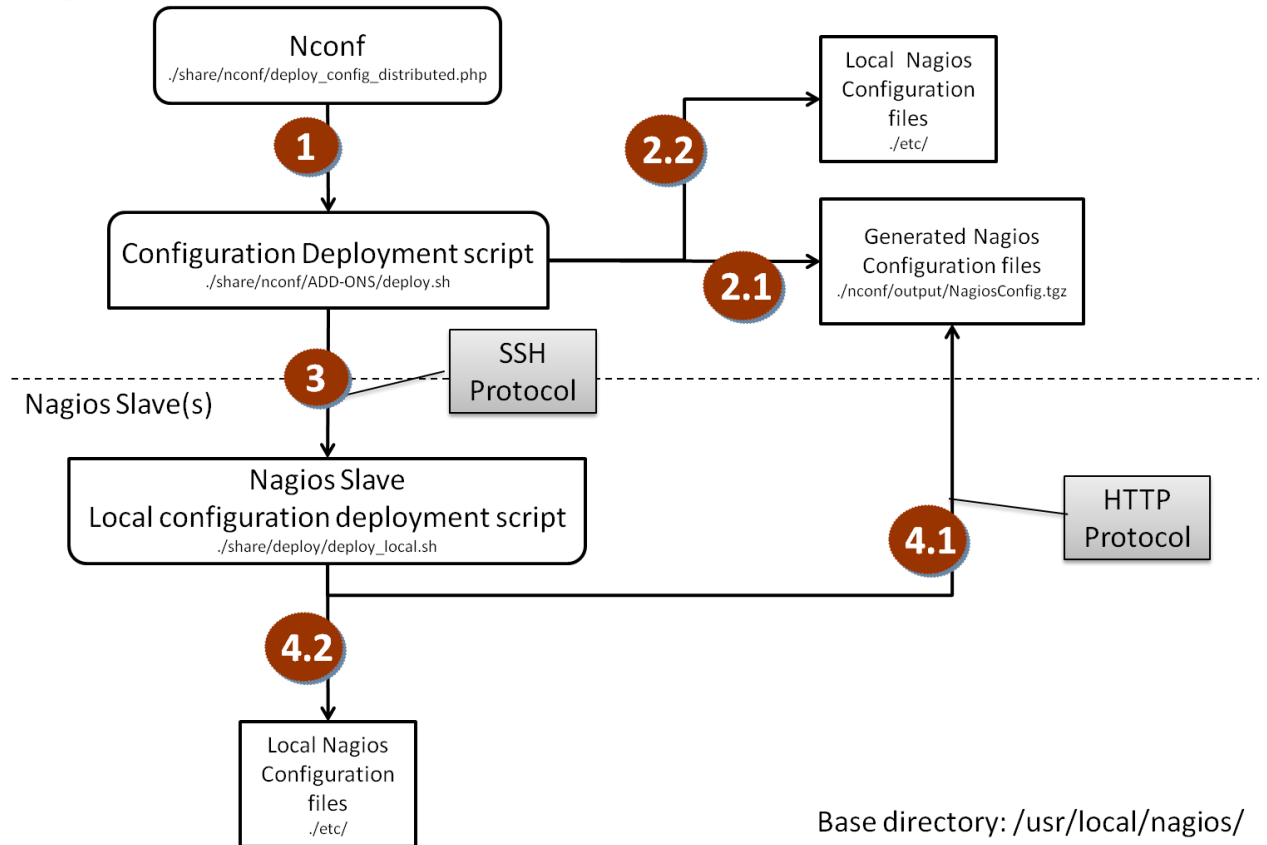


Figure 5.6.2 Nagios configuration deployment process

#### Nagios Master

1. The NConf deployment page starts the configuration deployment script.
2. Deploy the configuration local:
  - 2.1. The generated configuration is extracted to a temporary directory.
  - 2.2. The extracted configuration is copied to the local Nagios instance configuration directory and Nagios is ordered to reload its configuration.

#### Nagios Slave

3. The Nagios slave is called, using the SSH protocol, and is instructed to run its own deployment script.
4. Deploy the configuration on the slave:
  - 4.1. Download the generated configuration from the NConf server using the HTTP protocol. The configuration is extracted to a temporary directory.
  - 4.2. The extracted configuration is copied to the Nagios slave configuration directory and the Nagios slave is ordered to reload its configuration.

The steps for the Nagios slave are repeated for each Nagios slave.

Note: Which Nagios slaves must be configured is defined in the deployment script on the NConf server. (/usr/local/nagios/share/nconf/ADD-ONS/deploy.sh)

## 5.7 Monitoring Cordys with Nagios

This chapter describes how the Use Cases described in chapter 5.2 (Success Use Cases) were realized.

### 5.7.1 UC1: Web gateway failure

In this Use Case, the web gateway of a Cordys cluster must be monitored, and the Platform Operator must be notified in case of errors. Monitoring the web gateway consists of monitoring the Apache web server, because Cordys is using Apache as a web gateway. Nagios provides a standard check plug-in to monitor web servers. This plug-in checks if a web server responds to a standard HTTP request. If a HTTP response is received, and the HTTP status code is OK, the plug-in will report that the web server is operating properly.

Since this Use Case is the first, this Use Case also proves if Nagios is functioning properly. Monitoring the Apache web server is done by creating a service check on the host where Apache is running. Creating the host definition is not covered here because it is quite obvious.

#### 5.7.1.1 Creating the check command

The first step in creating a new service check is creating a check command definition. Using a check command definition, administrators only need to specify the details of a check plug-in once. After that, the defined command can be easily assigned to a host.

The following figure shows the NConf screen where the check command properties are added.

**Add checkcommand**

check command name	check_http	*
check command line	\$USER1\$/check_http -I \$HOSTADDRESS\$ \$ARG1\$	*
command description	ARG1=Multiple options	
default command params	!	separated by "'"
amount of params	1	*

Submit Reset

Figure 5.7.1 The add check command interface

#### Explanation

Explanation:

check command name:	This is the name by which Nagios can identify the plug-in.		
check command line:	This is the actual command which will be executed by Nagios. Nagios allows the use of macro variables which will be filled upon plug-in execution.		
	\$USER1\$	The system path to the location of the check plug-ins: /usr/local/nagios/libexec	
	\$HOSTADDRESS\$	This is the IP address of the host on which the service check will run.	
	\$ARG1\$	This is an optional variable which can be filled by an administrator when configuring a service check. Using \$ARG(1-10)\$ variables, administrators can supply extra parameters to a check plug-in.	
command description:	This is a description, for the command parameters, which will be shown in the NConf interface when assigning the plug-in to a certain host.		



5.7.1.2 Assigning the check command to a host

The next step in monitoring the web server, is assigning the check command to a host, the host on which the web server is running. This is done by creating a service definition, which is showed in the following figures.

Services of srv-nl-crd34 AC

Add additional services to host:

check\_http

check\_http

check\_ldap

check\_nt

check\_pim\_engine\_tasks

check\_ping

check\_pop

check\_smtp

check\_soap\_processor

check\_ssh

check\_tcp

check\_udp

Add

Figure 5.7.2 The check command, created earlier, is added to the host

Modify service

service name	check_http	*
check command	check_http	*
assigned to host	srv-nl-crd34 AC	*
check period	24x7	*
notification period	24x7	*
contact groups	admins	
servicegroups	ac cc system-services-ac system-services-cc	

Command syntax:  
ARG1=Multiple options

params for check command

Submit

Reset

Figure 5.7.3 Service check properties

Explanation

service name:	This is the name by which the service check will be shown in the Nagios web interface.
params for check command	By supplying the parameter “ !-u /cordys/”, the check plug-in will monitor the Cordys web interface at http://srv-nl-crd34/cordys/.

### 5.7.1.3 Testing the service check

After generating and deploying the configuration, the Nagios slave at the Admin Cluster started monitoring the web server. It reported back all information about the status of the web server.

The Nagios web interface now shows the current status of the web server.


Service Status Details For All Hosts						
Host ↑↓	Service ↑↓	Status ↑↓	Last Check ↑↓	Duration ↑↓	Status Information	
<a href="#">srv-nl-crd34 AC</a>	 check_http	OK	12-04-2009 11:11:40	3d 0h 48m 34s	HTTP OK: HTTP/1.1 200 OK - 1561 bytes in 0.001 second response time	

Figure 5.7.4 The current status is OK

As step 2 in the user case scenario describes, a failure on the web gateway is created. This is done by shutting down the web server.

From the command shell:

```
service httpd stop
```

Now, Nagios detects that the web server is not operational. The web interface shows a red warning, and the administrator receives an email in the mailbox.


Service Status Details For All Hosts						
Host ↑↓	Service ↑↓	Status ↑↓	Last Check ↑↓	Duration ↑↓	Attempt ↑↓	Status Information
<a href="#">srv-nl-crd34 AC</a>	 check_http	CRITICAL	12-04-2009 11:28:35	0d 0h 7m 56s	1/1	Connection refused HTTP CRITICAL - Unable to open TCP socket

Figure 5.7.5 The web server status is CRITICAL

```
***** Nagios *****

Notification Type: PROBLEM

Service: check_http
Host: srv-nl-crd34 (Admin Cluster)
Address: 10.1.30.68
State: CRITICAL

Date/Time: Fri Dec 4 11:21:45 CET 2009

Additional Info:

Connection refused HTTP CRITICAL - Unable to open TCP socket
```

Figure 5.7.6 The email which is received by the administrator

### 5.7.1.4 The results

Since Nagios detects the failure, and alerts the administrator, as shown in the previous paragraph, this Use Case was proved successful.

### 5.7.2 UC2: SOAP processor failure

The next step, in monitoring Cordys, is using SOAP to request the status of internal components. In this Use Case the SOAP processors will be monitored.

Since Nagios has no plug-which provides SOAP support to monitor services, a custom plug-in was written.

#### 5.7.2.1 Plug-in language

The first decision that was made was which language to use, to write the plug-in. The Python scripting language was chosen because it is ideal for smaller sized scripts, and Cordys has experience using Python.

#### 5.7.2.2 Connecting to Cordys

The next step was connecting to Cordys using a SOAP call. Determining which method must be called was done by using fiddler (<http://www.fiddler2.com/>) while viewing the SOAP processors from within the Cordys interface. Fiddler analyzes http traffic from browser applications.






















Monitor SOAP Processors - system									
List By		-All Computers-	Refresh Status Rate		15	sec	Show	Reload	
Name	Organiza	Compute	Startup T	Status	Process-J	SOAP Do	Processir	Last-time	
 Audit Service Process	system	srv-nl-cr	Automat	Started	30485	0	0	0	
 Business Process Man	system	srv-nl-cr	Automat	Started	29670	0	0	0	
 Classic Studio Proces:	system	srv-nl-cr	Automat	Started	29627	0	0	0	
 COBOC Processor	system	srv-nl-cr	Automat	Started	29829	0	0	0	
 Data Transformation I	system	srv-nl-cr	Automat	Started	29676	0	0	0	
 Email Processor	system	srv-nl-cr	Automat	Started	29711	0	0	0	
 Event Processor	system	srv-nl-cr	Automat	Started	29572	5	2059	38	
 Java Call Processor	system	srv-nl-cr	Automat	Started	29797	9	20313	10	
 LDAP Processor	system	srv-nl-cr	Automat	Started	29434	2332	279843	2	
 Metering Calculation F	system	srv-nl-cr	Automat	Started	29747	0	0	0	
 Metering Processor	system	srv-nl-cr	Automat	Started	29630	0	0	0	
 monitor@srv-nl-crd34	system	srv-nl-cr	Automat	Started	29367	140	763172	4	
 Notification Processor	system	srv-nl-cr	Automat	Started	29661	0	0	0	
 Provisioning Processo	system	srv-nl-cr	Automat	Started	31042	0	0	0	
 Rule Repository Proce	system	srv-nl-cr	Automat	Started	30487	0	0	0	
 Scheduler Processor	system	srv-nl-cr	Automat	Started	29655	0	0	0	
 Security Administratic	system	srv-nl-cr	Automat	Started	29579	0	0	0	
 Single Sign-On Servic	system	srv-nl-cr	Automat	Started	29638	17	5939	3	
 XForms Processor	system	srv-nl-cr	Automat	Started	29700	1	449	449	
 XForms Translation Pr	system	srv-nl-cr	Automat	Started	29609	0	0	0	
 XML Store Processor	system	srv-nl-cr	Automat	Started	29672	129	13462	3	
Please right-click on the grid header and select 'Column Chooser' to					Total 21	Running 21	Idle 0	Down 0	

Figure 5.7.7 The Cordys interface for monitoring the SOAP processors

The method that must be called is:

```
<List xmlns="http://schemas.cordys.com/1.0/monitor"/>
```

This method returns a list of the SOAP processors, including their current status.

This was first tested using SoapUI, a tool to send SOAP calls (<http://www.soapui.org/>).

While trying to call the method, it turned out that Cordys requires every SOAP call to be authenticated. This is shown in the returned SOAP error message (trimmed for readability).

```
<SOAP:Fault>
  <faultstring>
    Anonymous access is denied for the method 'List'.
  </faultstring>
</SOAP:Fault>
```

### 5.7.2.3 Authenticating a SOAP call

It is required to authenticate a SOAP call. This is done by calling the Single Sign-On processor which has the `Request` method. This method returns a SAML assertion and a signature, which must be supplied as a header in every SOAP call.

Using SoapUI it is possible to authenticate using SOAP. Then, after manually placing the returned SAML assertion and signature in a new SOAP call, it is possible to call the `List` method.

### 5.7.2.4 Writing the plug-in

Now the process of sending a SOAP call is successfully tested, by hand, with an existing tool, it was time to create the Nagios plug-in which will do the same, but automated. The mechanism of authenticating using a separated SOAP call, and using the received assertion and signature, will be automated in the script. The script must run without user input. The information required for authentication will be supplied as command line arguments to the plug-in. This is done by Nagios.

The script uses a number of steps to retrieve the status of a SOAP processor. The plug-in was created in the following steps:

1. Determine which SOAP processor must be monitored and the username and password for authentication.  
This information is supplied as parameters to the script.
2. Send the authentication SOAP call.
3. Filter the SAML assertion and the Signature from the answer.
4. Create a new SOAP call to the `List` method and insert the SAML assertion and the Signature from step 3 as a SOAP header.
5. Send the SOAP call to the `List` method.
6. Filter the SOAP answer for the requested SOAP processor.
7. Determine the status of the SOAP processor by reading the `status` field.
8. Return the status to the standard output, which will be read by Nagios.
9. Exit the script with the error code corresponding to the status.  
These codes are the same as described in chapter 5.5.1.1 (OCSP).

### 5.7.2.5 Using the plug-in

In order to use the plug-in the same steps as with UC1 were required.

- Create the check command definition  
Because the script requires several parameters, NConf is configured to ask for these parameters.
- Create service definitions  
For each SOAP processor that is monitored, a service definition was created.

### 5.7.2.6 Testing the service checks

After generating and deploying the new configuration, Nagios is monitoring a number of SOAP processors, as shown in the following figure.

Service ↑↓	Status ↑↓	Last Check ↑↓	Duration ↑↓	Status Information
<a href="#">Business Process Management</a>	PASV OK	12-04-2009 15:38:18	0d 0h 5m 59s	SOAP Processor OK (Business Process Management), status: started
<a href="#">LDAP Processor</a>	PASV OK	12-04-2009 15:38:38	0d 0h 5m 39s	SOAP Processor OK (LDAP Processor), status: started
<a href="#">check_http</a>	PASV OK	12-04-2009 15:38:38	0d 2h 56m 42s	HTTP OK: HTTP/1.1 200 OK - 1561 bytes in 0.001 second response time
<a href="#">monitor@srv-nl-crd34</a>	PASV OK	12-04-2009 15:38:48	0d 0h 5m 29s	SOAP Processor OK (monitor@srv-nl-crd34), status: started

Figure 5.7.8 The status of the SOAP processors is OK

As step 2 in the user case scenario describes, a failure on a SOAP processor is created. This is done by stopping a SOAP processor. In this case we test the Business Process Management processor. From the Cordys interface, stop is selected.

Name	Organizational Unit	Computer	Startup Type	Status	Process-ID
Audit Service Processor	system	srv-nl-cr	Automat	Started	30485
Business Process Management		-cr	Automat	Started	29670
Classic Studio		-cr	Automat	Started	29627
COBOC Processor		-cr	Automat	Started	29829
...		...	...	...	...

Figure 5.7.9 The BPM processor is ordered to stop

As shown, Nagios has detected the created failure and shows an alert.

Service ↑↓	Status ↑↓	Last Check ↑↓	Duration ↑↓	Status Information
<a href="#">Business Process Management</a>	PASV CRITICAL	12-04-2009 15:48:18	0d 0h 0m 9s	SOAP Processor CRITICAL (Business Process Management), status: not running
<a href="#">LDAP Processor</a>	PASV OK	12-04-2009 15:47:38	0d 0h 14m 49s	SOAP Processor OK (LDAP Processor), status: started
<a href="#">check_http</a>	PASV OK	12-04-2009 15:47:38	0d 3h 5m 52s	HTTP OK: HTTP/1.1 200 OK - 1561 bytes in 0.001 second response time
<a href="#">monitor@srv-nl-crd34</a>	PASV OK	12-04-2009 15:47:48	0d 0h 14m 39s	SOAP Processor OK (monitor@srv-nl-crd34), status: started

Figure 5.7.10 The status of the BPM processor is CRITICAL

### 5.7.2.7 The results

Since Nagios has detected the failure and has alerted the administrator, this Use Case was proven successful.

### 5.7.3 UC3: Engine task in 'waiting' state on Admin Cluster caused by error on Customer Cluster

In this Use Case the provisioning process will be monitored. This is the most complicated Use Case because information from multiple Cordys clusters must be combined.

#### 5.7.3.1 Problem description

In the provisioning process, some parts of the process take place on the Admin cluster, and some parts on the Customer cluster. If a step in a provisioning process requires action on the Customer cluster, the Admin cluster gives the instructions to the Customer cluster using a synchronous SOAP call.

If a problem occurs during this phase, it is detectable because the Admin cluster will get a timeout or a similar error.

If no problems occur, the Customer cluster will start executing the received instructions as a BPM. On the Admin cluster, the provisioning ticket corresponding to the current provisioning process is put in waiting state.

Under normal conditions, the Customer cluster will report back to the Admin cluster when it is finished. If reporting back is successful, the Admin cluster continues the provisioning process.

If reporting back fails, this failure is undetectable from the Admin cluster.

#### In short

The purpose of this Use Case is to detect situations where the Customer cluster has failed to report back to the Admin cluster.

#### 5.7.3.2 The designed solution

A design with proposals for two scenarios for monitoring this problem was created. The scenarios take a different approach.

In the first scenario, Nagios will collect information on the Admin cluster, send it to Nagios at the Customer cluster and thus combine information from the Admin cluster with information on the Customer cluster. This combined information is sent back to the Admin cluster where Cordys is updated with the combined information. This scenario requires a few changes in the SDF software.

In the second scenario, only information on the Customer cluster will be monitored in order to detect the described problem. This scenario requires changes to be made to Nagios in order to create a full solution for the problem. This is because Nagios cannot compare the information on the Customer cluster to other information and therefore it must have its own bookkeeping of acknowledged errors and unimportant information.

#### Decision

The first scenario was chosen because it gives the best usable solution. Additionally, making changes into Nagios is undesirable, and the modifications into SDF are very small and have zero impact on other functionality of SDF.

See the attached UC3 design for more detailed information about the scenarios.

### 5.7.3.3 The solution procedure

The solutions components and procedure steps are shown in the following figure.

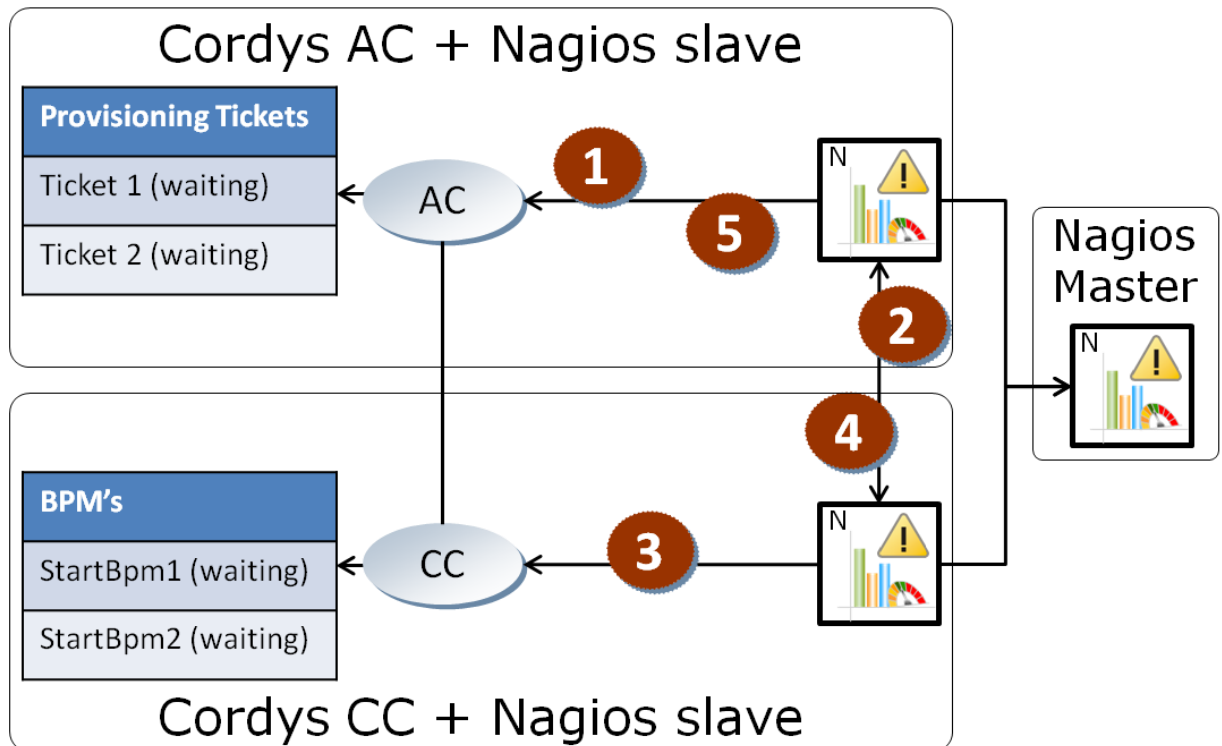


Figure 5.7.11 The diagram of the components and steps involved in the Use Case solution.

The procedure consists of the following steps;

On the Admin cluster:

1. Nagios requests a list of provisioning tickets in waiting state from the Provisioning Processor.
2. Nagios sends the list to the Nagios instance at the Customer cluster.

On the Customer cluster:

3. Nagios reads the received list of tickets and requests a list of Process Instances (BPM's), containing the `guids` from the received list, from the PIM (Process Instance Manager).
4. Nagios compares the statuses of the Process Instances to the statuses of the tickets and puts the mismatching Tickets, including their related Process Instances, in a list. Nagios sends this list to the Nagios instance at the Admin cluster.

Again on the Admin cluster:

5. Nagios reads the received list and orders the Provisioning Processor to put the tickets into error status.

As visible in the above list, the entire procedure consists of three parts. Therefore also three Nagios plug-ins were created:

- Request ticket list & send to customer cluster (AC)
- Compare ticket list to process instance list (CC)
- Update the tickets in the returned list (AC)

#### 5.7.3.4 Writing the plug-ins

##### Plug-in 1: distribute engine tasks list

The reason this name was chosen for this plug-in is pretty obvious, this plug-in itself doesn't actually do any monitoring. But it is necessary for the other plug-ins to function properly.

A lot of code could be reused from the SOAP processor plug-in:

- Parsing command-line arguments.
- Authentication module.
- Sending a SOAP call to Cordys.
- Parsing XML data.
- Filtering XML objects.

The plug-in was created in the following steps:

1. Read the command-line arguments:  
username and password to connect to Cordys.
2. Authenticate (using the SAML module from the SOAP processor plug-in).
3. Send the SOAP call to the Provisioning processor using the `GetPreviousEngineTasks` method. A filter is added to this request using the `Filter` element. Tickets which are already in `Error` state are filtered out.
4. Parse the received ticket list into a XML DOM structure.
5. Write the XML DOM structure to a file.
6. Copy the file to the Customer cluster using the `scp` command.
  - a. If copying failed, exit with status `CRITICAL`.
  - b. If copying succeeded, exit with status `OK`.



**Plug-in 2: check pim engine tasks**

The name describes that this plug-in compares the PIM with the Engine Tasks in order to check the consistency between them. The list of Engine Tasks which was received from the Admin cluster is compared to the PIM.

A lot of code could be reused from the SOAP processor plug-in, and from plug-in 2 from this Use Case:

- Parsing command-line arguments.
- Authentication module.
- Sending a SOAP call to Cordys.
- Parsing XML data.
- Filtering XML objects.
- Writing output to a file.

The plug-in was created in the following steps:

1. Read the command-line arguments:
  - Username and password to connect to Cordys.
  - The filename of the ticket file received from the Admin cluster.
  - Filename of the file to write the output to.
  - Server address of the Admin cluster
  - Username and password to connect to the Admin cluster
  - Location to upload the return file on the Admin cluster
2. Read the ticket file received from the Admin cluster
3. Authenticate (using the SAML module from the SOAP processor plug-in).
4. Send the SOAP call to the Business Process Management processor using the `QueryAdminData` method.
5. Browse the list of Process Instances and for each Process Instance:
  - Compare the status with the related ticket
  - If the statuses mismatch, add the ticket to the export list
6. Write the export list to the output file
7. Copy the output file to the Admin cluster using the `scp` command.
  - a. If copying failed, exit with status CRITICAL.
  - b. If copying succeeded
    - i. If one or more mismatches were found, exit with status CRITICAL.
    - ii. If no mismatches were found, exit with status OK.

**Plug-in 3: check process cc pim errors**

The name of the plug-in describes that this plug-in processes the list of PIM errors from the Customer cluster. Almost all code from the previous plug-in could be reused.

The plug-in was created in the following steps:

1. Read the command-line arguments:
  - Username and password to connect to Cordys.
  - The filename of the ticket error file received from the Customer cluster.
2. Read the ticket error file received from the Customer cluster.
3. Authenticate to Cordys.
4. Browse the list of error tickets, and for each ticket:
  - Create an error message with error details from the ticket file.
  - Send a SOAP call to the Provisioning processor using the `ActivityError` method. This method updates the ticket with the supplied error information.
5. Exit the plug-in:
  - a. If no tickets were updated, exit the script with status OK.
  - b. If one or more tickets were updated, exit the script with status CRITICAL.

**5.7.3.5 Using the plug-ins**

In order to use the plug-ins, configuring them into Nagios was required.

- Create the check command definition  
Because the plug-ins require several parameters, NConf is configured to ask for these parameters.
- Create service definitions  
Plug-in 1 and 3 were configured to run on the Admin cluster. Plug-in 2 was configured to run on the Customer cluster.

### 5.7.3.6 Testing the service checks

After generating and deploying the new configuration, Nagios is monitoring the provisioning process, consisting of 3 service checks. This is shown in the following figure.

Host	Service	Status	Last Check	Status Information
<a href="#">srv-nl-crd34 AC</a>	<a href="#">Distribute Engine Tasks List</a>	OK	12-08-2009 14:02:53	Writing local file... done. Start copying to Nagios slave... success!
	<a href="#">Process CC PIM Errors</a>	OK	12-08-2009 14:03:03	Reading ticket return file... done. File empty. No tickets to update.
<a href="#">srv-nl-crd88 CC</a>	<a href="#">Check PIM Engine tasks</a>	OK	12-08-2009 14:03:04	No tickets in ticket list. Reading PIM is skipped. Writing output file... done. Start copying to Nagios master... success!

Figure 5.7.12 The status of the Provisioning plug-ins is OK.

Step 4 in the Use Case dictates that a currently undetectable failure must be created. As explained in paragraph 5.7.3.1 (Problem description), this is a situation where the Customer cluster cannot report back to the Admin cluster in asynchronous calls.

The failure is created by modifying the `cluster.AdminClusterUri` property of the Provisioning processor on the Customer cluster so it points to a non-existing server.

Now, a new provisioning process is started. The tickets are shown in the following figure.

Overview of all tasks for the provisioning engine				
Close				
<input type="checkbox"/>	Creation date	Ticket state	Object type	Model name
<input type="checkbox"/>	8-12-2009 14:1...	Waiting	Order	101: Add account...
<input type="checkbox"/>	8-12-2009 14:1...	Waiting	UserApplSubscr	601: Add user ap...

Figure 5.7.13 The tickets are in waiting state.

Now, after the Customer cluster has stopped executing the BPM, Nagios detects the mismatch, and sends a list back to the Admin cluster. Meanwhile, an email was sent to inform the Administrator of the mismatch. The detection of the mismatch is shown in the following figure.






Host ↑↓	Service ↑↓	Status ↑↓	Last Check ↑↓	Status Information
<a href="#">srv-nl-crd34 AC</a> 	<a href="#">Distribute Engine Tasks List</a> 	OK	12-08-2009 14:21:53	Writing local file... done. Start copying to Nagios slave... success!
	<a href="#">Process CC PIM Errors</a> 	OK	12-08-2009 14:22:03	Reading ticket return file... done. File empty. No tickets to update.
<a href="#">srv-nl-crd88 CC</a> 	<a href="#">Check PIM Engine tasks</a> 	CRITICAL	12-08-2009 14:22:04	PIM MISMATCH, not all processes match to their ticket status. Mismatching tickets: 1 (Matching tickets: 0). Writing output file... done. Start copying to Nagios master... success!

Figure 5.7.14 Nagios at the Customer cluster has detected the mismatch, and sent this mismatch to the Admin cluster

Now, Nagios at the Admin cluster will read the received list with mismatching tickets, and will update the tickets in Cordys. As shown in the following figure, Nagios has received the list, and updated Cordys with the information. Additionally, an email has been sent to the Administrator informing him about the updated tickets.






Host ↑↓	Service ↑↓	Status ↑↓	Last Check ↑↓	Status Information
<a href="#">srv-nl-crd34 AC</a> 	<a href="#">Distribute Engine Tasks List</a> 	OK	12-08-2009 14:28:53	Writing local file... done. Start copying to Nagios slave... success!
	<a href="#">Process CC PIM Errors</a> 	CRITICAL	12-08-2009 14:29:13	Reading ticket return file... done. Requesting saml... done. Sending soap request... done. Tickets processed: 1
<a href="#">srv-nl-crd88 CC</a> 	<a href="#">Check PIM Engine tasks</a> 	CRITICAL	12-08-2009 14:29:04	PIM MISMATCH, not all processes match to their ticket status. Mismatching tickets: 1 (Matching tickets: 0). Writing output file... done. Start copying to Nagios master... success!

Figure 5.7.15 Nagios at the Admin cluster has processed the list of mismatching tickets, and updated Cordys.

The ticket is now updated. This is shown in the following figure.

Overview of all tasks for the provisioning engine			
X Close			
Creation date	Ticket state	Object type	Model name
8-12-2009 14:1...	Waiting	Order	101: Add account...
8-12-2009 14:1...	Error	UserApplSubscr	601: Add user ap...

Figure 5.7.16 The ticket was put into error state.

Note: The Order ticket will stay in waiting state. This is because the UserApplSubscr ticket is a child ticket of the Order ticket. It is designed this way, and this is a normal situation.

The error details show that the error is logged by the Nagios plug-in. This will enable administrators to distinguish errors from Nagios plug-ins from Cordys errors. The error information is shown in the following figure.

**(N)Error in BPM**

Back

Code **UCF0080\_Message\_wrapper**

Message **(N)Error in BPM**

Creation date **8-12-2009 14:29:08**

Details **com.cordys.ucf.base.exception.UcfException: (N)Error in BPM details:**

**Logged by Nagios plugin:**

**BPM: Ucf/StartBpm\_ucf30.bpm**

**BPM Status: ABORTED**

**BPM Error: <faults**

Figure 5.7.17 Screenshot of the error details, logged by the Nagios plug-in.

### 5.7.3.7 The results

The tests show that:

- Nagios has successfully detected the mismatch between the Process Instance on the Customer cluster and the Provisioning Ticket on the Admin cluster.
- Nagios has successfully updated the ticket to error state.
- Nagios has successfully notified the Administrator by email.

Therefore, the Use Case was proven successful.

## 6. Conclusions

---

### 6.1 The results

In drawing a conclusion, the purpose of the project is compared to the achievements.

The purpose of the project was to create an overview of the status the entire Multi-cluster Environment; to provide administrators with up-to-date information about the Cordys multi-cluster environment.

The created Conceptual Solution covers this concept adequately.

The testing environment has proved that monitoring Cordys components is possible and provides the missing information about the Cordys Multi-cluster Environment (See paragraph 5.7.1.4 about Use Case 1, paragraph 5.7.2.7 about Use Case 2 and paragraph 5.7.3.7 about Use Case 3).

This could be easily expanded to a solution where all Cordys components are monitored.

Additionally, this project has created knowledge about monitoring the Cordys system, and has helped to raise awareness about monitoring. This showed up during the many discussions and conversations. This awareness will help integrating the monitoring aspect into further application development.

### 6.2 The process

Overall the process went good. Indicators for this were: Results were delivered quick, enthusiastic reactions at the first presentation and positive feedback during the evaluation meetings.

The process also had a few weaknesses:

#### **Interviews**

A few things could have been done better in preparing the interviews described in chapter 4.2 (Requirements). The questions for the interview were not discussed before conducting the interviews. This resulted in discovering the incompleteness of the questions during the first interview.

This could have been avoided by reviewing the interview questions in advance.

#### **Designing the Conceptual Solution**

At the start of designing the Conceptual Solution, the following problem was encountered: Finding the most usable way of writing down the requirements took some time. This resulted in multiple formats and notations of the same requirements. This could have been avoided by deciding which one was going to be used, and then sticking to it. On the other hand, that might have led to a situation where requirements would have been written in an unusable or insufficient format.

One other problem was finishing the Conceptual Solution. When the design was almost finished, more ideas came up and it was not clearly defined when the Conceptual Solution would be finished. This resulted in perhaps spending too much time on the design. This could have been avoided by defining milestones for the Conceptual Solution, limiting the scope more, and more important: Defining the end.

#### **Writing the Thesis**

This was started too late. This was caused by planning insufficient time for it in the beginning of the project. This was not done because the plan of approach was not yet finished at that time, but it should have been done anyway to have initial thesis version ready earlier in the project. From this could be concluded that this internship was more about the work, than about graduating.

## 7. Recommendations

---

### 7.1 BPM vs. Provisioning Ticket comparison

#### 7.1.1 Defining matching conditions

**Situation**

In UC3, the status of Tickets is compared to the status of the corresponding Process Instance. The conditions on which the plug-in flags them as mismatching are hardcoded in the Nagios plug-in.

**Recommendation**

The matching conditions could be stored in a configurable matrix. This should be preferably configurable from NConf.

#### 7.1.2 Thresholds

**Situation**

If, for example, a ticket is in 'waiting' state, and its related Process Instance is also in 'waiting' or 'running' state, this is a matching situation according to Nagios. However, if it keeps this way for a week, it could be undesirable.

**Recommendation**

Enhance the Nagios plug-in so that it can handle time threshold and flag a situation as mismatching if a Process Instance is running or waiting too long.

#### 7.1.3 Provisioning models

**Situation**

Different provisioning models can be used, and each of them can involve different steps on the Customer cluster.

**Recommendation**

An analysis should be done if the solution to UC3 is suitable for all provisioning models. Additionally, different matching conditions could be desirable for each provisioning model. Therefore the Nagios plug-ins could be enhanced to distinguish the different provisioning models and apply different matching schemes to them.

### 7.2 Securing the communication between Nagios instances

#### 7.2.1 Users and Passwords

**Situation**

The root account is used in scripts for sending files to other Nagios instances.

**Recommendation**

Create a user account with limited access to only the file and directories necessary.

### 7.2.2 Configuration deployment

**Situation**

Slaves download their configuration from the NConf server using the HTTP protocol, and using HTTP basic authentication.

**Recommendation**

This could be changed to HTTPS or SCP.

### 7.2.3 Check results

**Situation**

NSCA is not configured to use authentication when sending check results to the Nagios master.

**Recommendation**

This could be configured to prevent 3<sup>rd</sup> party applications from sending false information to the Nagios master. NSCA can also be configured to use encryption.

Note: Authentication must be configured on both master and slave servers.

### 7.2.4 Ticket lists / Process Instance lists

**Situation**

Ticket lists and Process Instance lists are exchanged between the Nagios instances. This is done using SCP. This is a secure protocol.

**Recommendation**

Authentication is currently done using a username and password combination. This could be replaced with public key authentication.



## 7.3 Nagios implementation options

### 7.3.1 Nagios slave-master communication failure

#### Situation

In case of a communication error between the Customer Cluster and the Admin Cluster, Nagios slaves at the Customer Clusters will not be able to send their check results to the Nagios master. The Nagios master will detect this because it checks the age of check results. If check results are too old, Nagios will issue an alert, saying the service results are outdated. The maximum age is configurable. Nagios uses the term freshness for this topic. (See: [http://nagios.sourceforge.net/docs/3\\_0/freshness.html](http://nagios.sourceforge.net/docs/3_0/freshness.html) for more information).

The following figure shows an example of the Nagios master interface in this situation.

Service Status Details For All Hosts

Host	Service	Status	Last Check	Duration	Attempt	Status Information
srv-nl-crd34.AC	Business Process Management	CRITICAL	11-30-2009 15:29:02	0d 0h 1m 18s	1/1	CRITICAL: Service results are outdated! Check if the Nagios slave is operating properly.
	Email Processor	CRITICAL	11-30-2009 15:30:02	0d 0h 0m 18s	1/1	CRITICAL: Service results are outdated! Check if the Nagios slave is operating properly.
	LDAP Processor	CRITICAL	11-30-2009 15:30:02	0d 0h 0m 18s	1/1	CRITICAL: Service results are outdated! Check if the Nagios slave is operating properly.
	Metering Calculation Processor	CRITICAL	11-30-2009 15:30:02	0d 0h 0m 18s	1/1	CRITICAL: Service results are outdated! Check if the Nagios slave is operating properly.
	Metering Processor	CRITICAL	11-30-2009 15:29:02	0d 0h 1m 18s	1/1	CRITICAL: Service results are outdated! Check if the Nagios slave is operating properly.
	Notification Processor	CRITICAL	11-30-2009 15:29:02	0d 0h 1m 18s	1/1	CRITICAL: Service results are outdated! Check if the Nagios slave is operating properly.
	Provisioning Processor	CRITICAL	11-30-2009 15:29:02	0d 0h 1m 18s	1/1	CRITICAL: Service results are outdated! Check if the Nagios slave is operating properly.
	check_ifm	CRITICAL	11-30-2009 15:30:02	0d 0h 0m 18s	1/1	CRITICAL: Service results are outdated! Check if the Nagios slave is operating properly.
	check_http	CRITICAL	11-30-2009 15:30:02	0d 0h 0m 18s	1/1	CRITICAL: Service results are outdated! Check if the Nagios slave is operating properly.
	check_local_disk	CRITICAL	11-30-2009 15:29:02	0d 0h 1m 18s	1/1	CRITICAL: Service results are outdated! Check if the Nagios slave is operating properly.
	check_local_load	CRITICAL	11-30-2009 15:29:02	0d 0h 1m 18s	1/1	CRITICAL: Service results are outdated! Check if the Nagios slave is operating properly.
	check_local_procs	CRITICAL	11-30-2009 15:30:02	0d 0h 0m 18s	1/1	CRITICAL: Service results are outdated! Check if the Nagios slave is operating properly.
	check_local_swap	CRITICAL	11-30-2009 15:30:02	0d 0h 0m 18s	1/1	CRITICAL: Service results are outdated! Check if the Nagios slave is operating properly.
	check_local_users	CRITICAL	11-30-2009 15:29:02	0d 0h 1m 18s	1/1	CRITICAL: Service results are outdated! Check if the Nagios slave is operating properly.
	check_process_cc_pim_errors	CRITICAL	11-30-2009 15:29:02	0d 0h 1m 18s	1/1	CRITICAL: Service results are outdated! Check if the Nagios slave is operating properly.
	distribute_engine_tasks_list	CRITICAL	11-30-2009 15:30:02	0d 0h 0m 18s	1/1	CRITICAL: Service results are outdated! Check if the Nagios slave is operating properly.
	monitor@srv-nl-crd34	CRITICAL	11-30-2009 15:29:02	0d 0h 1m 18s	1/1	CRITICAL: Service results are outdated! Check if the Nagios slave is operating properly.
srv-nl-crd88.CC	Business Process Management	CRITICAL	11-30-2009 15:29:02	0d 0h 13m 18s	1/1	CRITICAL: Service results are outdated! Check if the Nagios slave is operating properly.
	Email Processor	CRITICAL	11-30-2009 15:29:02	0d 0h 13m 18s	1/1	CRITICAL: Service results are outdated! Check if the Nagios slave is operating properly.
	Notification Processor	CRITICAL	11-30-2009 15:29:02	0d 0h 13m 18s	1/1	CRITICAL: Service results are outdated! Check if the Nagios slave is operating properly.
	Provisioning Processor	CRITICAL	11-30-2009 15:29:02	0d 0h 13m 18s	1/1	CRITICAL: Service results are outdated! Check if the Nagios slave is operating properly.
	check_http	CRITICAL	11-30-2009 15:29:02	0d 0h 13m 18s	1/1	CRITICAL: Service results are outdated! Check if the Nagios slave is operating properly.
	check_local_disk	CRITICAL	11-30-2009 15:29:02	0d 0h 13m 18s	1/1	CRITICAL: Service results are outdated! Check if the Nagios slave is operating properly.
	check_local_load	CRITICAL	11-30-2009 15:29:02	0d 0h 13m 18s	1/1	CRITICAL: Service results are outdated! Check if the Nagios slave is operating properly.
	check_pim_engine_tasks	CRITICAL	11-30-2009 15:29:02	0d 0h 14m 18s	1/1	CRITICAL: Service results are outdated! Check if the Nagios slave is operating properly.
	check_ping	CRITICAL	11-30-2009 15:29:02	0d 0h 13m 18s	1/1	CRITICAL: Service results are outdated! Check if the Nagios slave is operating properly.
	check_ssh	CRITICAL	11-30-2009 15:29:02	0d 0h 13m 18s	1/1	CRITICAL: Service results are outdated! Check if the Nagios slave is operating properly.
	monitor@srv-nl-crd88	CRITICAL	11-30-2009 15:29:02	0d 0h 13m 18s	1/1	CRITICAL: Service results are outdated! Check if the Nagios slave is operating properly.

Figure 7.3.1 Example of the Nagios web interface while the Nagios slaves are down

Nagios doesn't do anything particular in this case.

#### Recommendation

It is possible to configure the Nagios master to take action in this case. For example to perform a ping to the Nagios slaves.

- This could be done using event handlers, which can be executed if a certain service fails repeatedly.
- Another option is to enhance the `stale_service` or `stale_host` script to investigate the error.

#### Links

The following pages contain more information about this topic:

- [http://nagios.sourceforge.net/docs/3\\_0/distributed.html](http://nagios.sourceforge.net/docs/3_0/distributed.html)
- [http://nagios.sourceforge.net/docs/3\\_0/freshness.html](http://nagios.sourceforge.net/docs/3_0/freshness.html)
- [http://nagios.sourceforge.net/docs/3\\_0/eventhandlers.html](http://nagios.sourceforge.net/docs/3_0/eventhandlers.html)