Building an Enterprise Nagios Framework

Building an Enterprise Nagios Framework

PUBLISHED BY:
Darren Hoch
hochdarren@gmail.com

More Technical Papers and Code Available at:

http://www.ufsdump.org

Table of Contents

1.0 Introduction

The goal of this paper is to provide a framework for deploying Nagios on an enterprise scale. This paper does not cover how to setup Nagios nor is it a "cookbook" for deploying Nagios. It describes how to leverage the features built into Nagios to build a scalable enterprise monitoring infrastructure.

There is ample documentation on the web on Nagios installation and configuration on the Nagios website (`http://www.nagios.org`). Within the website, there is a very simple quick start guide to setup a Nagios Management Server with minimal effort. (`http://nagios.sourceforge.net/docs/3_0/quickstart.html`).

Nagios also has many well written books (many used for this paper) that describe in detail how to setup and configure Nagios. These include:

- **"Nagios System and Network Monitoring" – Wolfgang Barth**
- **"Pro Nagios 2.0" – James Turnbull**
- **"Building a Monitoring Infrastructure with Nagios" – David Josephsen**

This paper does not cover the installation and configuration of Nagios. If you have not setup Nagios before, this paper may be of limited value.

1.1 Brief Overview of Nagios Concepts

The Nagios Management Server (NMS) daemon is called `nagios`. The NMS consists of a master daemon that executes checks on configured objects. Objects are containers of information that describe what, how often, and who should be notified about a host or service.

Object configurations are discussed later in this paper.

The (NMS) provides 4 main functions:

- **Check scheduling –** uses an algorithm (as opposed to a static timer like `cron`) to schedule the execution of checks on particular host or service
- **Check execution –** executes a check (a Nagios plugin), which is lightweight compiled binary or text script
- **Response handling –** process response codes from check execution and takes action based on response
- **Alerting –** generates and sends alerts to appropriate contacts or contact groups

The daemon process:

```
# ps -ef | grep nagios
nagios    2379     1  0 12:45 ?  00:00:06 /usr/sbin/nagios -d
/etc/nagios/nagios.cfg
```

The main configuration file is `nagios.cfg`.

```
# ls -l /etc/nagios/nagios.cfg
-rw-rw-r-- 1 nagios nagios 42773 Jun 13 17:33 /etc/nagios/nagios.cfg
```

The NMS configuration file specifies additional object configuration files that contain objects for the NMS to monitor:

```
# more /etc/nagios/nagios.cfg

<snip>

cfg_file=/etc/nagios/objects/sm_contacts.cfg
cfg_file=/etc/nagios/objects/sm_master_escalation.cfg
cfg_file=/etc/nagios/objects/sm_master_host.cfg
cfg_file=/etc/nagios/objects/sm_master_service.cfg
cfg_file=/etc/nagios/objects/sm_master_service_groups.cfg
cfg_file=/etc/nagios/objects/sm_master_time.cfg
cfg_file=/etc/nagios/check_nrpe_command.cfg
```

1.2 Nagios Client – NRPE Agent

The NRPE client runs on any system that the NMS monitors. The NRPE daemon listens for requests from the NMS to execute checks. Once it receives a request for a check, the NRPE daemon executes the system checks locally by executing the appropriate plugin and sending the result back to the Nagios server.

The NRPE daemon is not a requirement to run the Nagios. Without the NRPE daemon, Nagios will be limited to running checks on publicly exposed services (checking ports and banners, for example). The NRPE daemon allows for a NMS to check services that are not publicly and require local execution of the check.

The NRPE daemon on the client system:

```
# ps -ef | grep nrpe
nagios    1818     1  0 12:50 ?        00:00:00 /usr/sbin/nrpe -c
/etc/nagios/nrpe.cfg –d
```

The NRPE daemon executes a local check of a service by executing a plugin. The plugin execution is defined in the `/etc/nagios/nrpe.cfg` file:

```
# cat /etc/nagios/nrpe.cfg

<snip>

command[check_users]=/usr/lib/nagios/plugins/check_users -w 5 -c 10


<snip>
```

All of the plugins are located in `/usr/lib/nagios/plugins`:


```
# ls  /usr/lib/nagios/plugins/
check_apt       check_dns       check_ide_smart     check_log       check_ntp
check_real     check_swap   utils.pm
check_breeze    check_dummy     check_ifoperstatus  check_mailq     check_ntp_peer
check_rpc      check_tcp    utils.sh
check_by_ssh    check_file_age  check_ifstatus      check_mrtg      check_ntp_time
check_sensors  check_time

<snip>
```

Any plugin may be executed from the command line. This output is what is sent back to the NMS:

```
# /usr/lib/nagios/plugins/check_users -w 3 -c 10
USERS OK - 2 users currently logged in |users=2;3;10;0
```

The NRPE daemon runs on port 5666 by default and uses SSL (if compiled with SSL).

1.3 Nagios Objects

Nagios configuration is built on an object model. An object in Nagios is defined as a set of configuration parameters (hostname or IP address, for example). Nagios objects are defined in the `/etc/nagios/*.cfg` files on the Nagios Management Server.

There are many objects in Nagios:

- **Host objects –** defines the characteristics of a single host such as IP address and hostname
- **Host groups –** defines a series of hosts as a single object (`hostA` and `hostB` make up the "`Servers`" group, for example)
- **Services –** defines what and how to perform a check
- **Service groups –** defines a series of services to check at once
- **Contacts –** defines a person or people to contact and a means to contact them
- **Contact groups –** defines a series of contacts to alert
- **Commands –** defines how a check should be run (perform an SMTP check with a "`-H`" option, for example)
- **Escalation objects –** defines contact escalation paths if an alert is not addressed

1.4 Object Configuration Parameters

Each object in Nagios contains multiple configuration parameters. These parameters and standard name value pairs. Each object has a minimum set of these parameters.

The following example lists the minimum parameters required to define a host object.


```
define host{
        host_name acme-01
        alias Acme Appliance 01
        address 10.0.1.212
        }
```

The parameters are described below:

- `host_name` **–** the name of the system to be monitored
- `alias` **–** a more friendly name to be used for a management UI
- `address` – the IP address of the system to be monitored

1.5 Basic Object Configuration

A typical Nagios installation contains at least **1 host**, **contact** and **service** object along with a series of pre-defined command objects. The following are three objects monitor the amount of UNIX users on a host called `acme-01` and send both alerts for host and services to `dhoch@acme.com`

Monitor the host `acme-01`:

```
define host{
        use acme-host
        host_name acme-01
        alias Acme Appliance 01
        address 192.168.1.61
        }
```

Monitor the amount of UNIX users logged into the system. Notice that the configuration here is instructing the NMS to execute the check remotely via the NRPE configuration. The NMS is going to send a request to NRPE on the host `acme-01` to run the `check_users` command on `acme-01` and return the results.

```
define service{
        use acme-service-15m
        host_name acme-01
        service_description UNIX Users
        check_command   check_nrpe!check_users
        }
```

Send email alerts for both host and service problems to `dhoch@acme.com`**:**

```
define contact{
        contact_name                    acme
        alias                           Acme Customer
        service_notification_period     24x7
        host_notification_period        24x7
        service_notification_options    w,u,c
        host_notification_options       d,u
        service_notification_commands   notify-service-by-email
        host_notification_commands      notify-host-by-email
        email                           dhoch@acme.com
        }
```

1.6 Nagios Alerting

The NMS processes the responses back from the NRPE daemon. Based on the response code, the NMS determines whether or not to trigger an alert. If the NMS triggers an alert, it them determines who should receive the alert as defined by the contact object related to the service.

Alerting contacts based on the time the alert was generated is controlled by time objects. Using time objects to send alerts to different support groups is covered later in this paper.

1.6.0 Host Checks

The conducts a basic check on any host defined in an object. It uses the ICMP `ping` command to determine if a system is up and running.

If the NMS manages systems across the Internet and ping is disabled at a firewall, the NMS will generate an alert that the host is down.

The following are the host status response codes.

- **OK –** the host is up and available
- **DOWN –** the host is unavailable
- **UNREACHABLE –** the check never reached the host (intermediate network problem, for example)

1.6.1 Service Checks

The following are individual service check response codes:

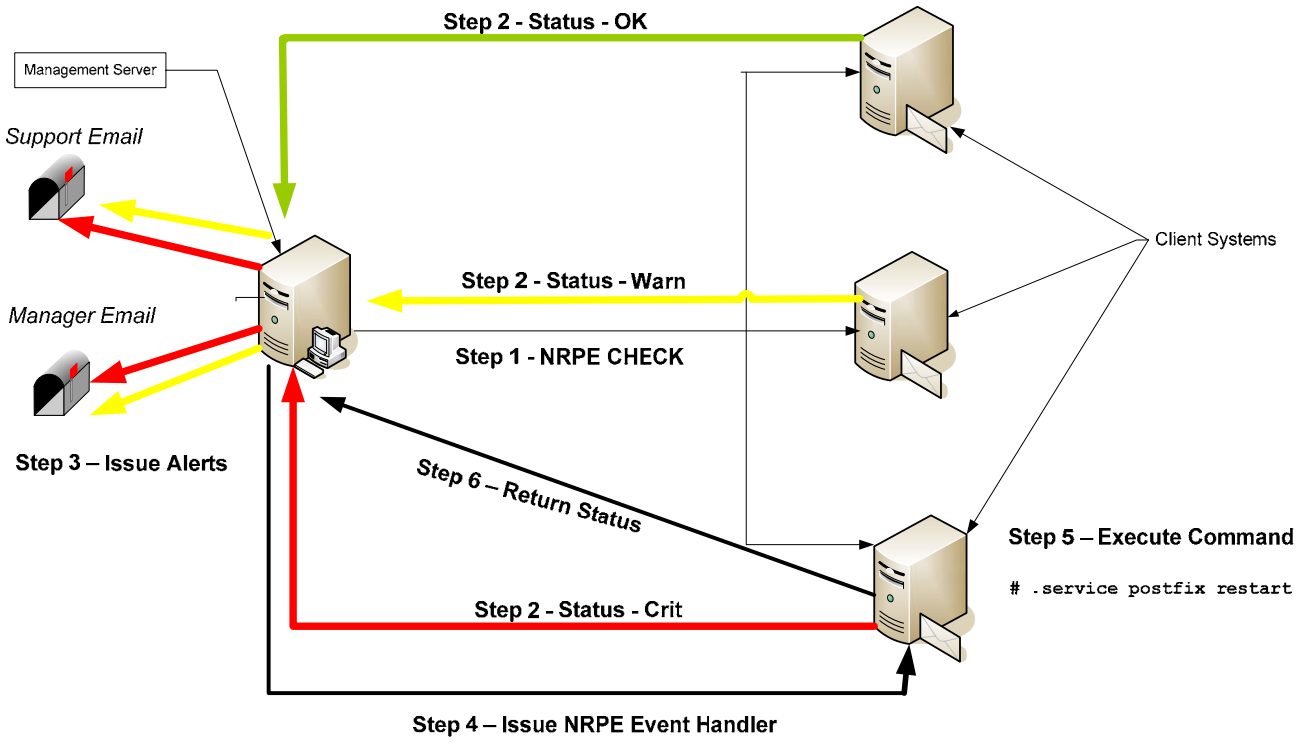
- **OK –** service is functioning normally
- **WARNING –** a service is reaching a user defined threshold (disk 85% full, for example)
- **CRITICAL –** a service has reached the top of a threshold or is down (disk is 100% full, for example)
- **UNKNOWN –** the NMS can't determine the state of the service due to a plugin error or malformed response

Nagios alerts may be sent based on any of the previously mentioned states in addition to some other conditions.

Nagios alerts may be issued on the following conditions:

- **Host unreachable, down or ok**
- **Service unknown, critical, warning or ok**
- **Service recovers from previous failed state**

Figure 1: Nagios Architecture



1.7 Nagios Web Management Interface

Nagios provides an optional web user interface (UI) to administer Nagios. The web UI has the following features:

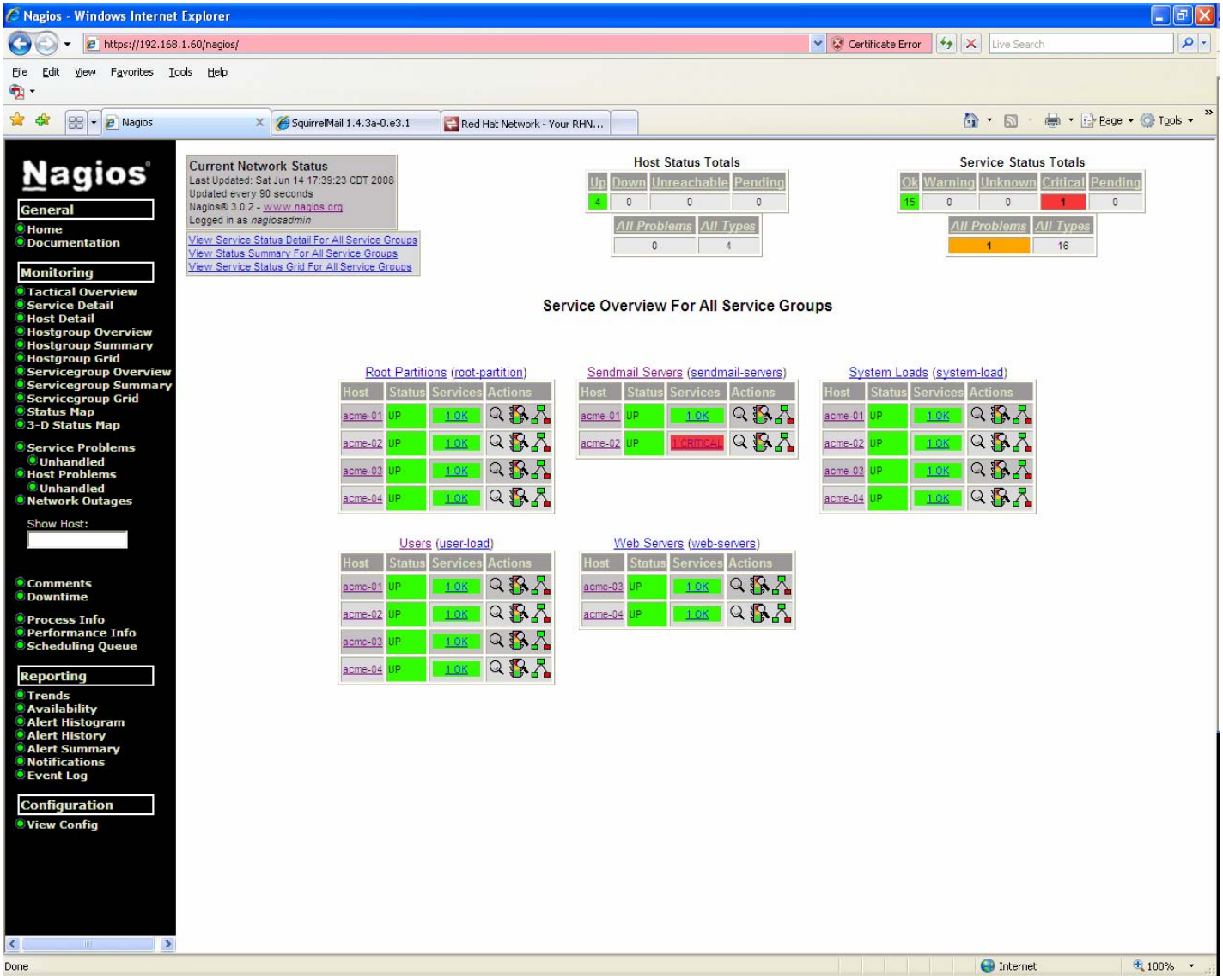
- **Real time reports of host and service status by individual hosts and services or host groups and service groups**
- **Contact group information**
- **Network topology graphs**
- **Log file viewing and analysis**
- **The ability to run "external commands" like enabling, disabling or restarting a check on a specific service.**

The Nagios Web UI is configured by default to run with the version of Apache that ships with Red Hat. It is possible to use any web server or configuration if needed. The Nagios Web UI supports HTTP authentication and SSL. These features are setup during the Nagios server installation.

The URL to the UI is located here:

```
https://nagios.acme.com/nagios
```

Figure 2: Nagios Web Interface



2.0 Enterprise Nagios Deployment

A default Nagios installation is suitable for a flat network of 1 – 10 server systems with one systems administrator. In order to scale Nagios to span multiple networks and groups, many of the features of the software need to be leveraged.

The following areas of the default installation need to be addressed when moving Nagios into the Enterprise:

- **Single contact for alerts 24x**7 – This does not accommodate a 2 or 3 shift workday with multiple contacts spread across multiple locations.
- **No ability to route alerts based on a grouping of hosts or systems –** What if different managers must be alerted for different systems in addition to the support staff?
- **No escalation alerts** – What if the primary oncall resource is not available?
- **Single host, service and alert configuration file** – All objects are organized around a single system. There are no global configurations across all systems.
- **False positives** – Network latency across a WAN often contributes to the generation of a false positive.

The rest of this paper describes how to create and enterprise scalable Nagios deployment. The following are the core building blocks of this architecture:

- **Global object creation**
- **Host/Service groupings**
- **Reducing false positives**
- **Building escalation paths**
- **Creating a master/slave setup**

This paper assumes that the reader is familiar enough with Nagios and is capable of installing and setting up the default configuration.

3.0 Enterprise Setup - Global Object Configuration

In order to effectively manage an infrastructure across many locations and networks, systems need to be grouped. For example, there may be a Sendmail cluster in data center A (DCA) and a cluster at the DR datacenter B (DCB). In addition to grouping systems based on location or function, different sets of administrators may be responsible for different groupings of systems. In a 24x7 operation, different groups of administrators working different shifts may also need to be alerted based on time.

In order to scale the monitoring of systems across data centers, administrators, and time periods, a series of "Global" configuration objects must be created. These global objects define the key components of the IT administration: what boxes, who monitors them, and when should they be monitored?

The following sections describe how to configure these objects based on a simulated IT monitoring environment for a company called "**Acme"** with the following attributes:

- **What boxes? –** a cluster of Apache and Sendmail servers
- **Who monitors them? –** There are two support teams. One resides in the US and one in India 24x5. A primary and secondary oncall engineer monitor the boxes over the weekends.
- **When should they be monitored? –** The US team monitors the infrastructure during US hours during the week. The India support team monitors the infrastructure all other hours during the week. The primary oncall engineer monitors the infrastructure Friday night until Sunday night. If the engineer does not respond, a secondary engineer receives alerts via an escalation path.

3.1 Global Object Templates

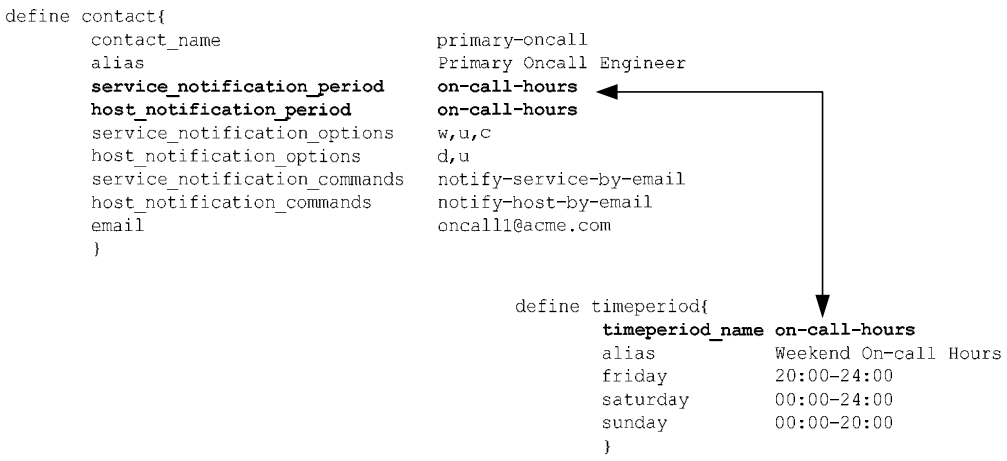
The success of a global object configuration works off of the concept of object templates. A template defines a list of values that apply to all objects in a configuration. Individual objects may then inherit these values by referencing the object template in their own template.

The following templates form the base of the global monitoring objects:

- **Master Time Template –** this template defines all of the time period objects required by all other objects
- **Master Contact Template –** this template defines the primary, secondary and fall back contacts
- **Master Host and Service Templates –** this template defines the global characteristics of host and service checking
- **Generic Host Group Template –** a generic template exists on the Nagios master server and is copied and modified to setup a new grouping of hosts and services that can be based on service type or location
- **Master Escalation Template –** this template defines the escalation paths when alerts are not addressed

In the following example, the object that defines the oncall engineer contact (`primary-oncall`) inherits the monitoring times from a global template object (`on-call-hours`).

Figure 3: Object Inheritance and Templates



For the examples used in this paper, the master object templates are stored in `/etc/nagios/objects.`

```
# ls /etc/nagios/objects/*master*
sm_master_escalation.cfg  sm_master_service.cfg
sm_master_time.cfg
sm_master_contacts.cfg  sm_master_host.cfg
sm_master_service_groups.cfg
```

3.2 Global Time and Contact Object Templates

The first step in defining a Nagios enterprise architecture is to figure out who will be alerted and when. The following sections will provide a global object configuration that supports routing of alerts to the appropriate IT staff. The overall goals for this architecture include:

- **Systems monitoring and alerting 24x7**
- **A 24x7 support staff split between the US and India**
- **Alert routing to the correct support staff based on their hours of operation**

3.2.0 Master Time Configuration

Time configuration is the most crucial part of the global object configuration. Time objects must be created for two discrete purposes: define when a system will be monitored and define when different groups receive alerts.

In the following example. the first object defines a 24x7 monitoring period that will be used for hosts and services:

```
# more /etc/nagios/objects/sm_master_time.cfg
define timeperiod{
        timeperiod_name 24x7
        alias           24 Hours A Day, 7 Days A Week
        sunday          00:00-24:00
        monday          00:00-24:00
        tuesday         00:00-24:00
        wednesday       00:00-24:00
        thursday        00:00-24:00
        friday          00:00-24:00
        saturday        00:00-24:00
        }
```

Optionally, if there are services that only need to be monitored during the work week (14x5), the object looks like this:

```
define timeperiod{
        timeperiod_name 14x5
        alias           14 Hours A Day, 5 Days A Week
        monday          06:00-20:00
        tuesday         06:00-20:00
        wednesday       06:00-20:00
        thursday        06:00-20:00
        friday          06:00-20:00
        }
```


The next set of time template objects define the time periods that the two different contact groups (US and India) work:

```
define timeperiod{
        timeperiod_name us-hours
        alias           US Working Hours
        monday          06:00-20:00
        tuesday         06:00-20:00
        wednesday       06:00-20:00
        thursday        06:00-20:00
        friday          06:00-20:00
        }

define timeperiod{
        timeperiod_name india-hours
        alias           India Working Hours
        sunday          20:01-24:00
        monday          00:00-05:59,20:01-24:00
        tuesday         00:00-05:59,20:01-24:00
        wednesday       00:00-05:59,20:01-24:00
        thursday        00:00-05:59,20:01-24:00
        friday          00:00-05:59
        }
```

The last object defines when the time period for the primary oncall engineer:

```
define timeperiod{
        timeperiod_name on-call-hours
        alias           Weekend On-call Hours
        friday          20:01-24:00
        saturday        00:00-24:00
        sunday          00:00-20:00
        }
```

3.2.1 Master Contact Configuration

Now that the time object templates are present, we define the contact groups. The contact groups inherit their monitoring times from the global time templates.

- **Primary on call engineer**
- **Secondary on call engineer**
- **India support contact**
- **US support contact**

The most important parameter to note in the master contact templates is the `service_notification_period` and `host_notification_period`. These define when a specific contact should receive alerts.

The first object defines the contact information for the US based support group. The US team uses an email alias called `support@acme.com` to communicate and inherits the time template `us-hours` from the global time template file.

```
# more /etc/nagios/sm_master_contacts.cfg
define contact{
        contact_name                    us-technical-support
        alias                           US Support Alias
        service_notification_period     us-hours
        host_notification_period        us-hours
        service_notification_options    w,u,c
        host_notification_options       d,u
        service_notification_commands   notify-service-by-email
        host_notification_commands      notify-host-by-email
        email                           support@acme.com
        }
```

The next two objects define the primary and secondary oncall engineers for the weekends. Both also pick up their time periods from the global time objects.

```
define contact{
        contact_name                    primary-oncall
        alias                           Primary Oncall Engineer
        service_notification_period     on-call-hours
        host_notification_period        on-call-hours
        service_notification_options    w,u,c
        host_notification_options       d,u
        service_notification_commands   notify-service-by-email
        host_notification_commands      notify-host-by-email
        email                           oncall1@acme.com
        }

define contact{
        contact_name                    secondary-oncall
        alias                           Secondary Oncall Engineer
        service_notification_period     on-call-hours
        host_notification_period        on-call-hours
        service_notification_options    w,u,c
        host_notification_options       d,u
        service_notification_commands   notify-service-by-email
        host_notification_commands      notify-host-by-email
        email                           oncall2@acme.com
        }
```

The fourth contact group is the India based support team who, like the others, inherit their time from the global time objects.


```
define contact{
        contact_name                    india-team
        alias                           India Support Team
        service_notification_period     india-hours
        host_notification_period        india-hours
        service_notification_options    w,u,c
        host_notification_options       d,u
        service_notification_commands   notify-service-by-email
        host_notification_commands      notify-host-by-email
        email                           indiateam@acme.com
        }                             bob@example.com
        }
```

3.2.2 Master Host and Service Configuration

The next step in creating an enterprise Nagios monitoring infrastructure is to define global objects for both hosts and systems. There are a generic set of actions Nagios should take across all hosts and services. These include when and how often to monitor and when to send an alert.

The following subsections define how to accomplish the following goals for hosts and services:

- **Enable 24x7 and 14x7 monitoring**
- **Enable 15 minute and 24 hour checks**
- **Determine how much time to wait before considering a host/service to be down**

3.2.2.0 Global Host Configuration Objects

Not every enterprise host needs to be monitored 24x7. Some hosts need monitoring during business hours only. In order to accomplish this in Nagios, there needs to be a total of 3 objects:

- **Generic 24x7 host monitoring object**
- **24x7 notification object**
- **14x5 notification object**

Note the difference between monitoring and notification. Nagios will monitor the hosts 24x7, but will only send notifications if the alert falls within the defined notification window.

In order to build two different notification objects, we must first define a generic monitoring object for both notification objects to inherit.

The following is a generic 24x7 monitoring object:

```
# more /etc/nagios/sm_master_host.cfg
define host{
      name                            generic-host
      active_checks_enabled           1
      passive_checks_enabled          0
      check_freshness                 0
      freshness_threshold             600
      notifications_enabled           1
      event_handler_enabled           1
      flap_detection_enabled          1
      failure_prediction_enabled      1
      process_perf_data               1
      retain_status_information       1
      retain_nonstatus_information    1
      check_period                    24x7
      register                        0
        }
```

The most important configuration parameter here is the `check_period` parameter with of value of `24x7`. This value is inherited from the `sm_master_time.cfg` file discussed earlier.

The next two objects inherit the values from the generic host (`generic-host`) object and define when to send alert notifications via the `notification_period` parameter:

```
define host{
        name                            notify-24x7
        use                             generic-host
        max_check_attempts              30
        check_command                   check-host-alive
        notification_period             24x7
        notification_interval           15
        notification_options            d,u
        register                        0
        }


define host{
        name                            notify-14x5
        use                             generic-host
        max_check_attempts              15
        check_command                   check-host-alive
        notification_period             14x5
        notification_interval           15
        notification_options            d,u
        register                        0
        }
```


Both of these objects inherit the 24x7 monitoring object. The difference lies in the notification objects. Individual host objects inherit either the **`notify-14x5`** (non-mission critical) or the **`notify-24x7`** (mission critical) templates based on the monitoring SLA for that host.

Figure 4: Template object inheritance of the `notify-14x5` template host object

3.2.2.1 Global Service Configuration Objects

The global configuration of services is exactly identical to the configuration of hosts in terms of a 24x7 and a 14x5 notification period. Some services are mission critical and others are not. In addition to **when** services are checked, Nagios offers the **frequency** in which services are checked.

The parameter that checks frequency is called `normal_check_interval`.

Global service template objects start with a generic service object:

```
# more /etc/nagios/objects/sm_master_service.cfg
define service{
        name                            generic-service
        active_checks_enabled           1
        passive_checks_enabled          1
        parallelize_check               1
        obsess_over_service             1
        check_freshness                 0
        notifications_enabled           1
        event_handler_enabled           1
        flap_detection_enabled          1
        failure_prediction_enabled      1
        process_perf_data               1
        retain_status_information       1
        retain_nonstatus_information    1
        is_volatile                     0
        register                        0
}
```

Mission critical services need to be monitored 24x7 and checked as frequently as possible. Non-mission critical services can be monitored 24x5 and checked less frequently. The following sets of objects define both a 15 minute interval and 24 hour interval object. Both inherit the `generic-service` object.

The first object is the mission critical object for a service. It checks every 15 minutes and monitors/alerts 24x7.

```
define service{
      name                            15min-24x7
      use                             generic-service
      check_period                    24x7
      max_check_attempts              3
      normal_check_interval           15
      retry_check_interval            5
      notification_options            u,c
      notification_interval           15
      notification_period             24x7
      register                        0

         }
```

The following object checks a service every 15 minutes 24x7 but only notifies 14x5.

```
define service{
        name                            15min-14x5
        use                             generic-service
        check_period                    24x7
        max_check_attempts              3
        normal_check_interval           15
        retry_check_interval            5
        notification_options            u,c
        notification_interval           15
        notification_period             14x5
        register                        0
        }
```

The next object is defined for non-mission critical systems. The object checks a service once a day and sends notifications 14x5. The only value that changes is the `normal_check_interval` (in minutes) from `15` to `1440`.

```
define service{
        name                            24-14x5
        use                             generic-service
        check_period                    24x7
        max_check_attempts              1
        normal_check_interval           1440
        retry_check_interval            15
        notification_options            u,c
        notification_interval           15
        notification_period             14x5
        register                        0
        }
```


Figure 5: Template object inheritance for a 15 minute checks 14x5 of a service

3.3 Global Object Review

All global objects have been successfully configured to align Nagios with the IT organization's 24x7 support staff. The global objects define the following behaviors:

- **All hosts and services are monitored 24x7**
- **Nagios monitors hosts continuously and services at either 15 minute or 24 hour intervals depending on the service**
- **Nagios sends alerts on mission critical services 24x7 and non-mission critical services 14x5**
- **Alerts are routed to the primary on call during US hours and the India support team during all other hours**

4.0 Individual Host and Service Objects

After the completion of the global object configuration, it is now possible to create individual host and service objects. Enterprise infrastructures may have hundreds of systems and thousands of service to monitor. These resources are often grouped by a specific task (web servers, routers, databases, etc). Therefore, it would make sense to group host and service objects by function.

The following sections group hosts and services to accomplish the following organizational goals:

- **Add an additional contact to receive alerts per each group (a manager of the mail servers, for example)**
- **Group hosts and services by functions (web servers, mail servers, etc)**
- **Check services across all hosts on either 15 minute or 24 hour intervals**
- **Present a monitoring view in the Nagios Web UI of all services across hosts**

The following sections will outline how to create a grouping of web servers and sendmail servers.

4.1 Grouping Hosts and Services

Nagios object configurations may be maintained in any number of configuration files. In a global organization, there are multiple IT managers and stakeholders. Most organizations have groups like network, systems, storage, etc. Each one of these groups may have a different management structure.

The first step in grouping hosts and services is to create one configuration file per grouping. In the following example, the systems, hosts, and contacts are grouped by type of service: email (`email.cfg`) and web server (`web-server.cfg`). Each set of services are managed by a different manager or key IT stakeholder.

4.1.0 Creating a Contact Group

The first part of each of these files contains an additional contact object. This contact object enables you to define a manager or IT stakeholder for just the group of hosts and services without modifying the global object templates.

The following object defines the contact information for the manager of the email server cluster.


```
# more /etc/nagios/services/email.cfg
define contact{
      contact_name                    Manager
      alias                           Email Server Manager
      service_notification_period     24x7
      host_notification_period        24x7
      service_notification_options    w,u,c
      host_notification_options       d,u
      service_notification_commands   notify-by-email
        host_notification_commands      host-notify-by-email
        email                           emailmanager@acme.com
        }
```

We have declared that Manager will receive alerts 24x7 for any hosts or systems in the email group. In order to add the manager to the global contact list, he or she must be included in a contact group for the email group.

The next object to be defined in the email configuration group is the contact group who should receive alerts for the email servers.

```
 define contactgroup{
        contactgroup_name   email-group
        alias               Email Group Contacts
        members            us-support,India-support,primary-oncall,Manager
        }
```

In order for Manager to receive alerts for just the email systems, generic host and service templates must be configured that define the email contact group. The actual hosts and services will import the values in these generic objects.

The following is an example of a generic email host template:

```
define host{
      name                            email-host
      use                             notify-24x7
      contact_groups                  email-group
      register                        0
        }
```

Likewise, there is a service object that defines how often the service should be checked. Since email is mission critical, this service will be checked every 15 minutes 24x7:

```
define service{
        name                            email-service-15m
        use                             15min-24x7
        contact_groups                  email-group
        register                        0
        }
```


In the previous example, we have built a contact group that now includes Manager. We have also included the India team and the primary on call. It is important to note how this individual contact group will relate to the global contact and alerting configuration. If Nagios generates an alert for any email systems, it will be routed the following ways:

- **Manager – all alerts all the time**
- **primary-oncall – on weekends**
- **support-team – only during US based hours**
- **India-team – only during India hours**

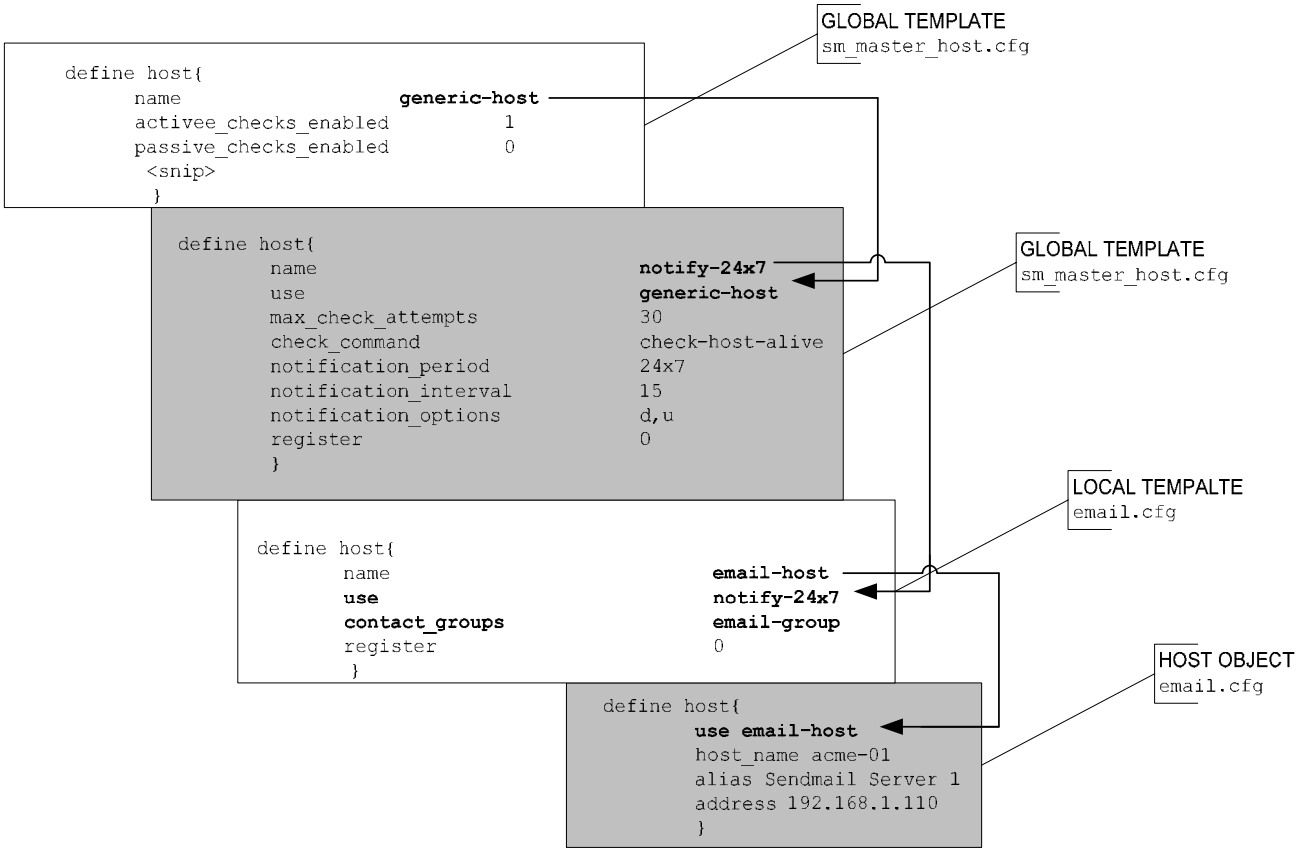
The previous template examples can easily be modified to create a grouping of other servers (web or mail, for example).

4.1.1 Defining a Host

The actual hosts to be monitored are now defined in the group configuration file. The following example defines two individual database hosts called `acme-01` and `acme-02`. You can add as many hosts as needed in the configuration file.

```
define host{
        use email-host
        host_name acme-01
        alias Sendmail Server 1
        address 192.168.1.110
        }

define host{
        use email-host
        host_name acme-02
        alias Sendmail Server 2
        address 192.168.1.111
        }
```


Figure 6: Individual host object template inheritance



4.1.2 Define a Service

The last step in this process is defining the actual service to be checked. This step is rather simple at this point given that most of the configuration has been completed. Services may be monitored at the host or host group level. In an enterprise infrastructure, it is far more scalable to create groups of many hosts and then monitor by group as opposed to defining individual host systems.

The following object defines the two previously defined host systems into a single host group called `email-systems`:

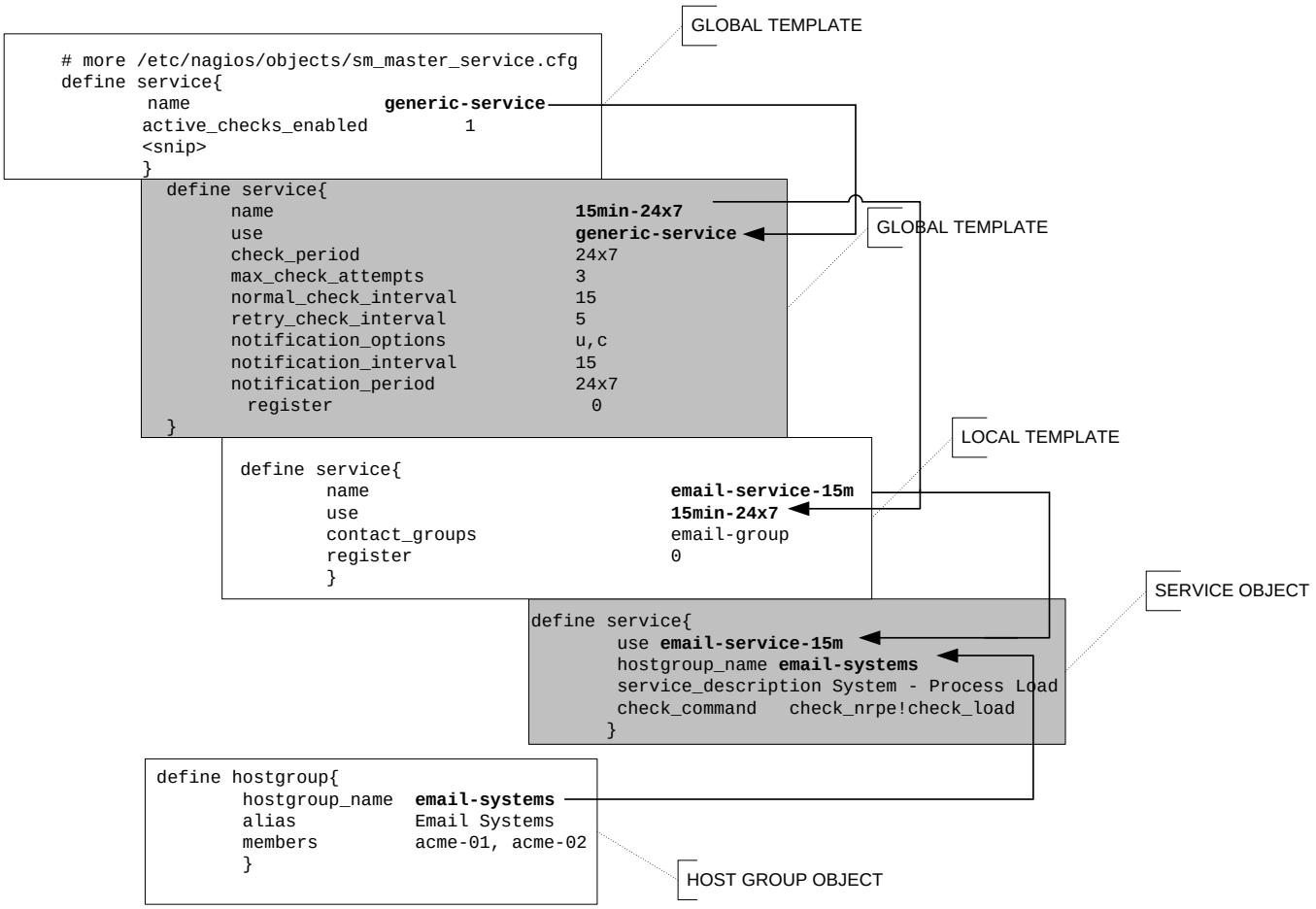
```
define hostgroup{
        hostgroup_name  email-systems
        alias           Email Systems
        members         acme-01, acme-02
        }
```

In the following example, two service checks define using the NRPE framework. The first checks the status of a disk on 24 hour intervals and the second checks system load every 15 minutes.

```
define service{
        use email-service-24
        hostgroup_name email-systems
        service_description System - Disk
        check_command   check_nrpe!check_disk
        }

define service{
        use email-service-15m
        hostgroup_name email-systems
        service_description System - Process Load
        check_command   check_nrpe!check_load
       }
```

The following figure demonstrates the relationship of the service objects.

Figure 7: Individual service object inheritance template



4.2 Individual Host and Service Review

Now that both global and individual objects are completely setup, the following goals have been accomplished:

- **Create logical groupings of hosts/services across the enterprise**
- **Enable both alerts for global support staff and specific stakeholders in a technology grouping**
- **Ability to easily add and remove both contacts, hosts per service grouping**

5.0 Reducing False Positives

The default Nagios configuration is tuned for a flat LAN based network of hosts and services. This configuration does not take into account network latency, scheduled down time, and host dependencies. For example, the NMS may need to check 15 hosts behind a router. If the router itself is unreachable, the NMS will flood the system with alert messages stating that all 15 hosts are down when in reality, just the router is down.

The following are 2 methods to reduce the amount of false positives in an enterprise configuration of Nagios: tuning check intervals and defining service dependencies.

5.1 Tuning Check Intervals

In large distributed networks there are many instances when a host or service may be temporarily unavailable. Some examples include: a network route drops, a system panics and reboots, significant network latency, a system administrator accidentally brings a system into single user mode.

Tuning the check intervals down to account for these variances in global networks may reduce some of these false positives.

There are 4 main parameters that define when a service should be checked:

- **`max_check_attempts –`** Defines how many times the NMS should check on a service (soft failure) before considering the service critical (hard failure).
- **`normal_check_interval –`** Defines the interval period between checks when the NMS receives and **`OK`** from the service.
- **`retry_check_interval –`** Defines the interval period between checks when the NMS receives a **`CRITICAL`** from the service.
- **`notification_interval –`** Defines how often the named contact or contact group should receive an alert while the NMS receives **`CRITICAL`** messages.

The following example settings allow for up to 15 minute outages of a service and will send alerts every 15 minutes until the service recovers from critical.

```
max_check_attempts              3
normal_check_interval           15
retry_check_interval            5
notification_interval           15
```

In addition to lengthening the time to allow for blips, reducing the notifications to just `UNREACHABLE` and `CRITICAL` will also cut down on alerts. In a 24x7 environment, any `CRITICAL` message will receive administrator attention to resolve the issue. Therefore, it is implied the service will eventually recover. Hence the RECOVER notification is implied and does not need to be an alert:

```
notification_options            u,c
```

5.2 Service Dependencies

Service dependencies suppress active checking and notification of services dependent on other services. For example, a web server and a database server deliver a web application. If the state of either service is unknown or critical, then the whole web application is offline. There is no sense in checking either service until the service that generated the alert is fixed.

When a service enters into an `UNKNOWN` state, it is possible that connectivity to a server has been dropped. As a result, it is possible that the NMS will generate a storm of UNKNOWN alert messages for each service. Using service dependencies cuts down on these unnecessary notifications.

In the following example, the NMS stops checking and notifying for the SMTP servers if the HTTP servers are in an `UNKNOWN` or `CRITICAL` state.

```
define serviceescalation{
      hostgroup_name                email-servers
      service_description           *
      dependent_servicegroup_name   web-servers
      dependent_service_descritpion *
      execution_failure_criteria    u,c
      notification_failure_criteria w,u,c
}
```

Host dependencies also exist for Nagios, as well as the `parent` directive. The `parent` directive helps determine which hop in a network is down, but does not suppress notifications or alerts.

6.0 Master Escalation Configuration

In the event that a primary on call engineer (`oncall1`) does not respond to an alert sent to Nagios, an escalation must occur. Nagios supports multiple layers of escalations. In the following example, we have defined an escalation path that alerts a secondary on call (`oncall2`) and tertiary in the event that a primary on call engineer does not respond to a service alert.

```
# more /etc/nagios/sm_master_escalation.cfg

# The secondary oncall receives an alert message if 20 minutes
# has lapsed and the service is still critical.

define serviceescalation{
        host_name               *
        service_description     *
        first_notification      2
        last_notification       3
        notification_interval   20
        contact_groups          secondary-oncall-group
        }

# The support alias receives an alert message if 60 minutes
```

```
# has lapsed and the service is still critical.

define serviceescalation{
        host_name               *
        service_description     *
        first_notification      3
        last_notification       3
        notification_interval   20
        contact_groups          support-alias
        }
```

The important parameters here are `first_notification` and `last_notification`. These describe when an escalation should occur. The secondary oncall engineer should receive an alert the second time the NMS sends an alert message that a service is down. Given the sample configuration used in this guide, that escalation should occur roughly 15 minutes after the first alert. In other words, the primary oncall engineer has 15 minutes to fix the problem and return the service to the state OK before the secondary oncall engineer will be notified.

The following is a sample of the mail log file on the NMS. By looking at the log file time stamps, we can see that the NMS sent an alert to the secondary oncall engineer due to the fact that the state of the service was still `CRITICAL`.

```
Jun 14 16:56:01 ng-server sendmail[7217]: m5ELu1uX007215:
to=<oncall1@acme.com>, ctladdr=<nagios@localhost.localdomain> (500/500),
delay=00:00:00, xdelay=00:00:00, mailer=esmtp, pri=120599, relay=acme.com.
[192.168.1.235], dsn=2.0.0, stat=Sent (Ok: queued as 967733A280)

Jun 14 17:19:35 ng-server sendmail[7437]: m5EMJZ1D007435:
to=<oncall2@acme.com>, ctladdr=<nagios@localhost.localdomain> (500/500),
delay=00:00:00, xdelay=00:00:00, mailer=esmtp, pri=120599, relay=acme.com.
[192.168.1.235], dsn=2.0.0, stat=Sent (Ok: queued as 279F73A280)
```

The escalation contacts described above use contact groups. This enables an administrator to allow for multiple recipients of a single escalation. The first escalation is sent to the `secondary-oncall-group`. This group is defined in the `sm_master_contact.cfg` file and consists of the secondary on call engineer.

```
define contact{
        contact_name                    secondary-oncall
        alias                           Secondary Oncall Engineer
        service_notification_period     oncall-hours
        host_notification_period        oncall-hours
        service_notification_options    w,u,c
        host_notification_options       d,u
        service_notification_commands   notify-by-email
        host_notification_commands      host-notify-by-email
        email                           oncall2@acme.com
        }

define contactgroup{
        contactgroup_name               secondary-oncall-group
        alias                           Secondary Oncall Person
```

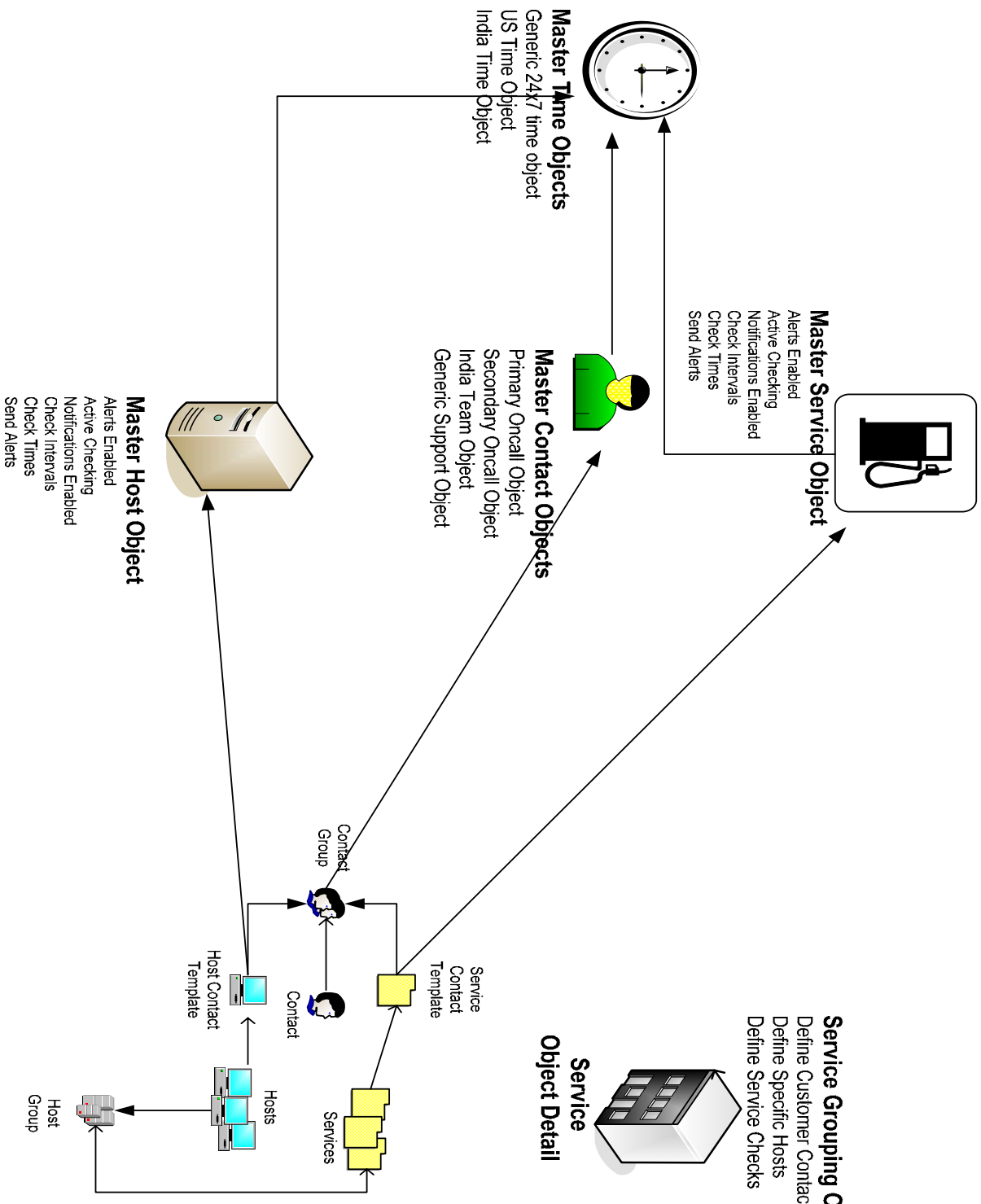


```
        members                         secondary-oncall
        }
```

The tertiary escalation goes straight to the US based support alias defined in the `sm_master_contact.cfg` file:

```
define contact{
        contact_name                    technical-support
        alias                           Support Alias
        service_notification_period     24x7
        host_notification_period        24x7
        service_notification_options    w,u,c
        host_notification_options       d,u
        service_notification_commands   notify-by-email
        host_notification_commands      host-notify-by-email
        email                           support@acme.com
        }

define contactgroup{
        contactgroup_name               support-alias
        alias                           support@example.com
        members                         technical-support
        }
```


Figure 8: Enterprise Nagios Object Configuration



7.0 Nagios Master and Slave Setup

The Nagios server maintains limited redundancy via a slave server. This redundancy is achieved by using the NSCA passive check system. NSCA functions in reverse of NRPE. It works as a proxy for Nagios results, forwarding them to one or many passive Nagios hosts. A passive Nagios host is defined as a host that does not execute active checks.

This paper uses this model to create a master and slave setup. The master server functions normally with one enhancement, the NSCA sending program. The master conducts standard NRPE checks on systems and receives the results. An additional global parameter is enabled to then forward the results (after recording them locally) to the Nagios slave server.

There is an NSCA listener bound on port 5667 on the slave server. The slave server interprets these as passive checks and has them enabled in the global Nagios configuration file. Since the active server is doing sufficient checking and forwarding results, the slave server has active checks and notifications disabled globally.

In order to monitor whether the master service is monitoring, the NRPE service runs on the slave server. NRPE contains one active check and that is to see if the Nagios process is running on the master server. As stated before, all customer checks/notifications are disabled.

In the event that the Nagios process on the master dies, the slave server will send an email notification to the administrator. The administrator must either restart the Nagios service on the master server or decide to elevate the slave server to master.

In order to do this, the following steps must take place:

1. Power off the Nagios master server.
2. Enable active checking in the master host/service templates on the slave server.
3. Reload the Nagios on the slave server.

7. 1 Limitations of Master/Slave Setup

The master/slave setup provides continuity of service at the sacrifice of historical data. The slave server contains all the same historical data as the master server due to the replication of the NSCA service on the master. If the master dies and the slave is elevated, the slave will continue to maintain an accurate history as it will now conduct active checks of all services.

The problem occurs when the master server is elevated back to master. At this point, the slave has maintained the latest historical data on customer hosts/services. There is no way currently to push this data back to the master server. When the master server resumes, it will be missing the historical data the slave collected when the master was offline.

7.2 Sending Passive Checks to the Slave

The NMS must be configured to send check results to the to the NSCA listener on the slave server. This is controlled by the `send_nsca.cfg` file (after the NSCA program has been installed). The only parameters that need to be modified are the password and encryption method. In the following example, the password will be used to encrypt results using 3DES.

```
# vi /etc/nagios/send_nsca.cfg

password=strongmail
encryption_method=3
```

In order to forward results to the Nagios slave, the master must "obsess" over service checking. This feature must be enabled. Once enabled, Nagios will execute the commands defined in the `ocsp_command` and `ochp_command`.

```
# vi /etc/nagios/nagios.cfg

<snip>

obsess_over_services=1
obsess_over_hosts=1

ocsp_command=send_service_check
ochp_command=send_host_check


<snip>
```

The administrator must create the `send_service_check` and `send_host_check` commands defined in the nagios.cfg. These are the commands that will send results to the slave server.

```
# vi /etc/nagios/nsca_commands.cfg

define command{
        command_name    send_service_check
        command_line    /usr/lib/nagios/plugins/send_service_check
$HOSTNAME$ '$SERVICEDESC$' $SERVICESTATEID$ '$SERVICEOUTPUT$'
        }


define command{
        command_name    send_host_check
        command_line    /usr/lib/nagios/plugins/send_host_check
$HOSTNAME$ $HOSTSTATEID$ '$HOSTOUTPUT$'
        }
```


As with all other config files, the NSCA command file must be added to the `nagios.cfg`.

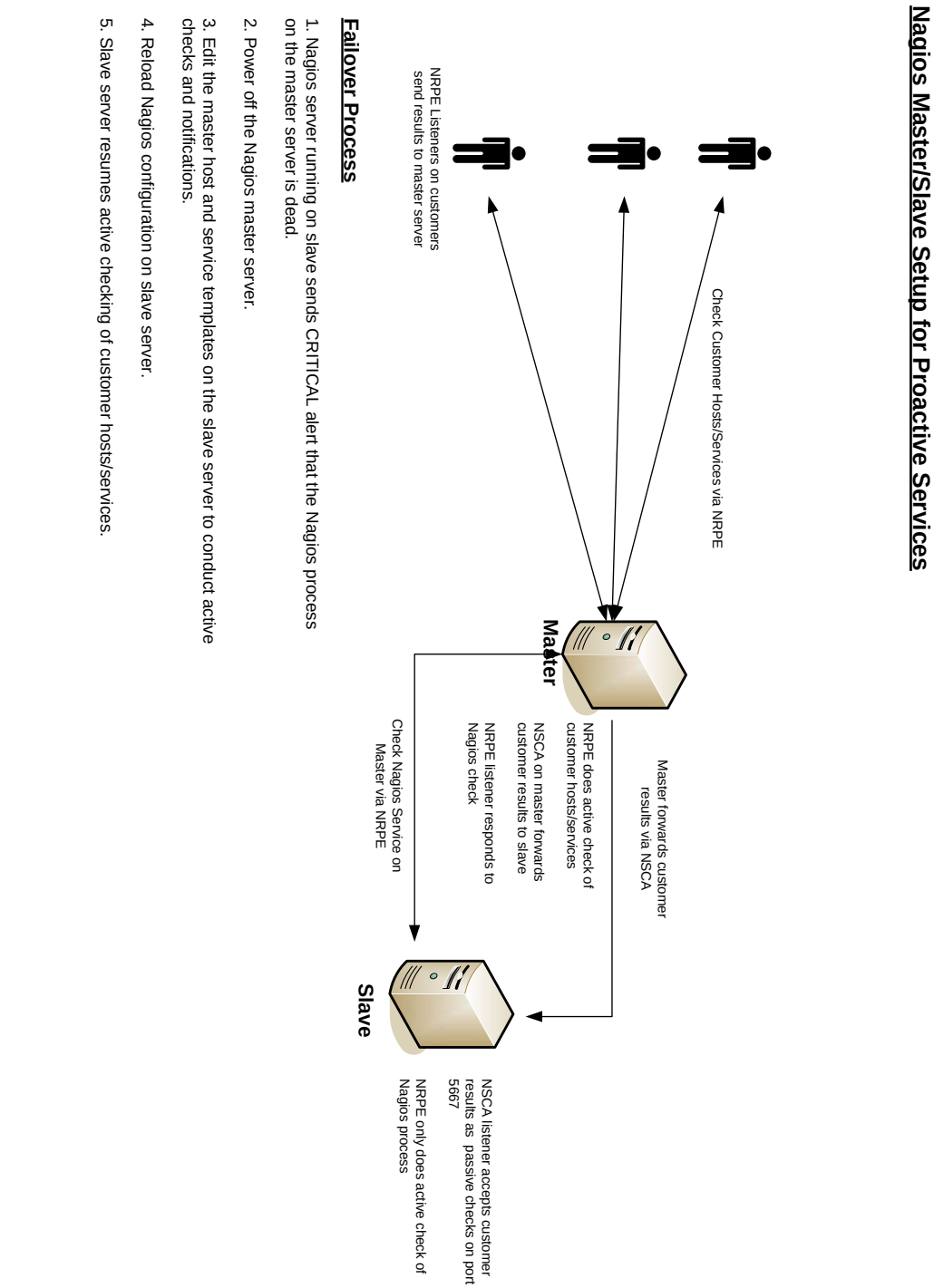
The `send_service_check` and `send_host_check plugins` must be created by the system administrator. Although defined in a command file, these do not exist. The IP addresses of the slave server (`192.168.29.213`) are defined in the individual plugins.

```
# vi send_service_check
#!/bin/sh
/bin/echo "$1","$2","$3","$4" | /usr/bin/send_nsca -H 192.168.29.213 -p
5667 -c /etc/nagios/send_nsca.cfg -d ","


# vi send_host_check
#!/bin/sh

/bin/echo "$1","$2","$3" | /usr/bin/send_nsca -H 192.168.29.213 -p 5667
-c /etc/nagios/send_nsca.cfg -d ","
```

The slave server contains an exact copy of all objects on the master with one exception to the configuration. It must be configured to not conduct active checks and notifications. This will result in duplicate work of both systems. This requires a modification to the global host and service templates.

```
slave# more /etc/nagios/objects/sm_master_host.cfg
define host{
        name                            generic-host
        active_checks_enabled           0
        passive_checks_enabled          1
        check_freshness                 0
        freshness_threshold             600
        notifications_enabled           0
        event_handler_enabled           1
<snip>
      }

slave# more /etc/nagios/objects/sm_master_service.cfg
define service{
      name                              generic-service
      active_checks_enabled             0
      notifications_enabled             0
<snip>
}
```

Figure 9: Master Slave Workflow



8.0 Conclusion

The previous sections of this paper describe how to organize the features of Nagios in a way that scales to enterprise deployments. By following the principles outlined in this paper, a systems administrator should now be able to configure Nagios in a way that spans multiple groups of hosts, systems, and administrators in a way that yields the most accurate information.

References

Barth, Wolfgang. **Nagios: System and Network Monitoring**. San Francisco, CA: No Starch Press, Inc.. 2006.

Josephsen, David. **Building a Monitoring Infrastructure with Nagios**. Boston, MA: Prentice Hall 2007.

Turnbull, James. **Pro Nagios 2.0**. Berkely, CA: Apress 2006.

