Contenido

[Introducción 3](#_Toc10472922)

[Antes de empezar 4](#_Toc10472923)

[Instalación de Nagios Core 4.0.4 5](#_Toc10472924)

[Instalando paquetes previos. 5](#_Toc10472925)

[Creando un usuario y grupo 5](#_Toc10472926)

[Instalando Nagios Core 6](#_Toc10472927)

[Instalando Nagios Plugin 6](#_Toc10472928)

[Configuración de la interfaz Web 7](#_Toc10472929)

[Instalación y configuración de clientes Linux 9](#_Toc10472930)

[Configurando el host 9](#_Toc10472931)

[Personalizando los comandos NRPE 10](#_Toc10472932)

[Comprobaciones desde Nagios 10](#_Toc10472933)

[Configurando Nagios 11](#_Toc10472934)

[Definiendo el host 11](#_Toc10472935)

[Definiendo el hostgroup 13](#_Toc10472936)

[Definiendo servicios en Linux con check\_nrpe 13](#_Toc10472937)

[Definiendo el comando 14](#_Toc10472938)

[Configurando y modificando las plantillas 14](#_Toc10472939)

[Modificando plantillas para los hosts 14](#_Toc10472940)

[Configurando las alertas vía correo 16](#_Toc10472941)

[Instalando y configurando el correo 16](#_Toc10472942)

[Definiendo contactos en Nagios 18](#_Toc10472943)

[Definiendo el comando para enviar notificaciones 19](#_Toc10472944)

[Comprobaciones 19](#_Toc10472945)

[Creando plugins de monitorización 20](#_Toc10472946)

[Creando Event-Handlers 22](#_Toc10472947)

[Instalación de InfluxDB y configuración con Nagios (nagflux). 24](#_Toc10472948)

[Requisitos. 24](#_Toc10472949)

[Instalando InfluxDB 24](#_Toc10472950)

[Configuración de InfluxDB 24](#_Toc10472951)

[Instalación y configuración de Nagflux. 25](#_Toc10472952)

[Requisitos. 25](#_Toc10472953)

[Instalando Nagflux. 25](#_Toc10472954)

[Configurando Nagflux 26](#_Toc10472955)

[Comprobaciones 28](#_Toc10472956)

[Configuración y comandos de Nagios para enviar PerfData a InfluxDB. 28](#_Toc10472957)

# Introducción

En este documento explicaremos como instalar y configurar un servidor Nagios Core 4, un servidor Grafana 6.1 y un servidor InfluxDB 1.7, todos ellos con los plugin necesarios corriendo sobre los sistemas Debian 9 stretch y Raspbian 9 stretch, desde cero.

**¿Qué son y para qué sirven cada uno de los servidores?**

* **Nagios** es un sistema de monitorización de redes de código abierto ampliamente utilizado, que vigila los equipos (hardware) y *servicios* que se especifiquen, alertando cuando el comportamiento de los mismos no sea el deseado. Entre sus características principales figuran la monitorización de servicios de red (SMTP, POP3, HTTP, SNMP, etc.), la monitorización de los recursos de sistemas de hardware (carga del procesador, uso de los discos, memoria, estado de los puertos, etc.), independencia de sistemas operativos, posibilidad de monitorización remota mediante túneles SSL cifrados o SSH, y la posibilidad de programar plugin específicos para nuevos sistemas.  
  Aclarar aquí que el termino *servicio* se usa muy libremente en Nagios y puede referirse a servicios que corren en un host (POP, SMTP,HTTP…) o a algún otro tipo de métrica asociada a un host (respuesta a un ping, número de usuarios logueados en el sistema, espacio libre en un disco…)
* **Grafana** es una herramienta de código abierto para el análisis y visualización de métricas. Se utiliza frecuentemente para visualizar de una forma elegante series de datos en el análisis de infraestructuras y aplicaciones. Suele ir acompañada de otras aplicaciones/plugin que la complementan, en nuestro caso InfluxDB, para proporcionarle las fuentes de datos que alimentan las gráficas y cuadros de mando creados.
* **InfluxDB** es una base de datos basada en series de tiempo (time-series database), ideal para logs o datos de graficas que se generen en vivo (dashboards). Programado en go (Lenguaje Google) permite la interacción vía API HTTP(S)(JSON) e interfaz web y los datos se gestionan con un lenguaje similar a SQL. Este tipo de bases de datos ha experimentado un crecimiento exponencial en los últimos años, con la popularización de IoT, Big Data y otras tecnologías que recogen mucha información en el tiempo. Se necesita un sistema que maneje de forma eficiente esas series de datos, con miles de datos por segundo, y de los que necesitamos hacer cálculos y agregar información de manera eficiente, tales como medias, máximos, búsquedas en el tiempo, etc., y todo ello en tiempo real. Dentro de esta categoría, InfluxDB es una base de datos que supera los esquemas SQL y NoSQL, permitiendo hacer análisis de miríadas de datos en tiempo real.

## Antes de empezar

Para este proyecto se utilizará la siguiente estructura.

* Un Nagios Core 4.0.4 corriendo sobre una Raspberry PI 3 con sistema Raspbian (Debian) 9 stretch con DN suso.\*\*\*\*\*\*\*\*\*.com
* Un Grafana 6.1 corriendo sobre un portátil con Debian 9 stretch con DN davide.\*\*\*\*\*\*\*\*.com
* Una InfluxDB corriendo sobre una Raspberry PI 3 con sistema Raspbian (Debian) 9 stretch con DN moses\*\*\*\*\*.\*\*\*\*\*.org

El esquema de cómo se comunican entre ellos es el siguiente:

Operario N1

Nagios

Nagflux

InfluxDB

Grafana

Histou

# Instalación de Nagios Core 4.0.4

## Instalando paquetes previos.

Para asegurar el correcto funcionamiento de toda la infraestructura hemos de instalar los siguientes paquetes en el servidor de Nagios.

* Apache 2
* PHP (En nuestro caso la versión 7)
* GCC Compiler and Development libraries
* GD Development libraries.

El proceso de instalación de estos paquetes es sencillo.

Instalación de Apache 2:  
 | # apt-get install apache2

Instalación de PHP:  
 | # apt-get install php  
Deberemos asegurarnos que entre los paquetes instalados de php esté incluido “libapache2-mod-php”. De no ser así deberemos instalarlo manualmente.

Instalación de las librerías necesarias:  
 | # apt-get install make gcc g++ sudo libgd2-xpm libgd2-xpm-dev libpng12-dev libgd-tools libpng3-dev

## Creando un usuario y grupo

Como siempre en Linux, es mejor crear un usuario para cada servicio, de manera que si por algún motivo un atacante llegara a acceder al sistema a través de Nagios, solamente afectaría al usuario de Nagios y solo tendría los permisos de este usuario.   
Creamos el usuario:

| # adduser nagios

Tras crear el usuario procederemos a crear el grupo (aunque el grupo se crea de manera automática al crear el usuario), y le añadimos a los usuarios nagios y www-data (servidor web).

| # groupadd nagios  
 | # usermod –G nagios nagios  
 | # usermod –G www-data, nagios www-data

## Instalando Nagios Core

Para instalar Nagios Core hemos de obtener la URL de la versión que se desee instalar, en nuestro caso la “*Latest stable release*”. Una vez obtenido descargamos, descomprimimos e instalamos Nagios con los siguientes comandos:  
  
 | # wget http://downloads.sourceforge.net/project/nagios/nagios -4.x/nagios- 4.0.4/nagios-  
 |4.0.4.tar.gz

| # tar xvzf nagios-4.0.4.tar.gz  
 | # cd nagios-4.0.4/  
 | # ./configure –prefix=/usr/local/nagios –with-cgiurl=/nagios/cgi-bin –with-htmlurl=/nagios/   
 |--with-nagios-user=nagios –with-nagios-group=nagios –with-command-group=nagios   
 | # make all  
 | # make install  
 | # make install-init  
 | # make install-commandmode  
 | # make install-config

Es importante que ejecutemos el siguiente comando, ya que será el encargado de establecer Nagios en el inicio del sistema.  
  
 | # make install-daemoninit

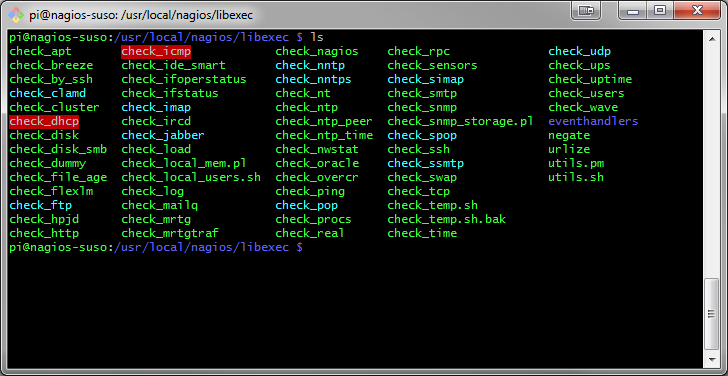
## Instalando Nagios Plugin

Nagios plugin es un añadido que nos permite conectar con servicios más específicos en caso de querer conectar con un host y saber su carga de CPU, RAM HDD y otros servicios o dispositivos.

Para la instalación hemos de obtener el URL del plugin que queremos instalar, para ello hemos de volver a la web de Nagios, pero a la sección de plugin. Ya con la URL ejecutamos los siguientes comandos en la terminal:

| # wget <http://nagios-plugins.org/download/nagios-plugins-2.0.tar.gz>  
 | # tar xvzf nagios-plugins-2.0.tar.gz  
 | # cd nagios-plugins-2.0/  
 | # ./configure  
 | # make && make install

Si todo ha ido correctamente deberemos ver los plugin en la carpeta */usr/local/nagios/libexec*



## Configuración de la interfaz Web

Ya tenemos instalado Nagios, ahora nos queda configurar la interfaz web en la que poder observar la monitorización de los servicios. Para ello crearemos un sitio en apache2:  
 | # nano /etc/apache2/sites-available/nagios.conf

Dentro añadimos el siguiente texto:  
   
<VirtualHost \*:80>  
 DocumentRoot /usr/local/nagios/share  
 ScriptAlias /nagios/cgi-bin /usr/local/nagios/sbin  
 <Directory “/usr/local/nagios/sbin”>  
 Options ExecCGI  
 AllowOverride None  
 Order allow,deny  
 Allow from all  
 AuthName “Nagios Access”  
 AuthType Basic  
 AuthUserFile /usr/local/nagios/etc/htpasswd.users  
 Require valid-user  
 </Directory>  
 Alias /nagios /usr/local/nagios/share  
 <Directory “/usr/local/nagios/share”>  
 Options None  
 AllowOverride None  
 Order allow,deny  
 Allow from all  
 AuthName “Nagios Access”  
 AuthType Basic  
 AuthUserFile /usr/local/nagios/etc/htpasswd.users  
 Require valid-user  
 </Directory>  
</VirtualHost>

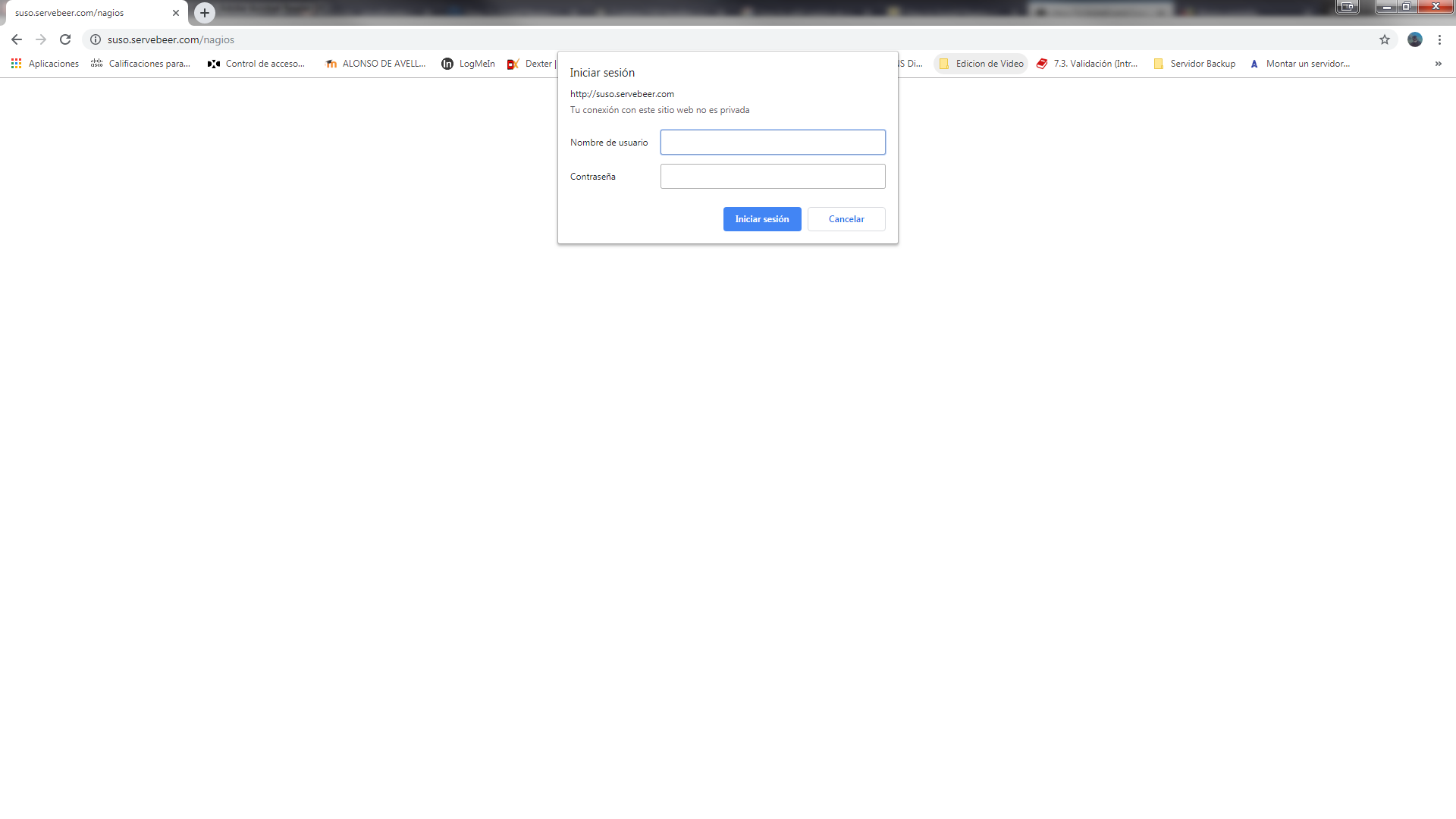
Como cada vez que se define un sitio nuevo en apache, hemos de activar el sitio y recargar el servicio de apache2.  
 | # a2ensite nagios.conf  
 | # systemctl reload apache2.service

A continuación creamos el fichero que será usado para almacenar las contraseñas de los usuarios con permisos para entrar a la interfaz web.   
 | # htpasswd –c /usr/local/nagios/etc/htpasswd.users nagiosadmin

Es importante que si vamos a añadir más usuarios al fichero no usemos el parámetro –c, este se ha de usar solo la primera vez para “c”rear el fichero. Si volvemos a ejecutar el comando con el parámetro lo que haremos será sobrescribir el fichero, perdiendo los usuarios que hubiera dentro.

Por último, en el fichero */usr/local/nagios/etc/cgi.cfg* hemos de cambiar la opción “use\_authentication=0” ha de estar en “use\_authentication=1”, por defecto viene a 1, pero es conveniente revisarlo.

Una vez efectuados los pasos anteriores hemos de reiniciar Apache2 y Nagios, para que todo coja efecto.  
 | # systemctl restart apache2  
 | # systemctl restart nagios

Si todo ha ido correctamente se nos solicitará autenticación en el momento de acceder a la ip del servidor de nagios desde un navegador.  


## Instalación y configuración de clientes Linux

En este punto vamos a ver como configurar equipos Linux para ser monitorizados.

Se detallará:

* Algunas de las distintas formas de conseguir monitorizar los equipos
* La utilidad de algunos ficheros y como modificarlos
* El uso de las plantillas para hosts y servicios
* Instalación del software necesario en los equipos.

### Configurando el host

Para realizar la monitorización del cliente debemos instalar en cada uno de los clientes que deseemos monitorizar el agente nrpe de nagios, el cual se encargará de establecer la comunicación entre el servidor de monitorización y el cliente mediante un puerto, que podremos especificar. Mediante este agente podremos llamar, desde el servidor, a los diferentes plugins de monitorización del cliente, siendo esta, una herramienta muy potente, ya que nos permitirá crear nuestros propios plugins de manera sencilla.

Lo primero será actualizar la caché de los repositorios de nuestro sistema.  
 | # apt-cache search nrpe

Tras realizar esta operación podremos descargar e instalar la versión más actual del agente, para lo cual ejecutaremos el sig. comando:  
 | # apt-get install nagios-nrpe-server nagios-plugins-basic nagios-plugins

Con esto ya tendríamos el agente instalado en los equipos, solo nos quedaría activa el servicio e iniciarlo:   
 | # systemctl enable nagios-nrpe-server  
 | # systemctl start nagios-nrpe-server

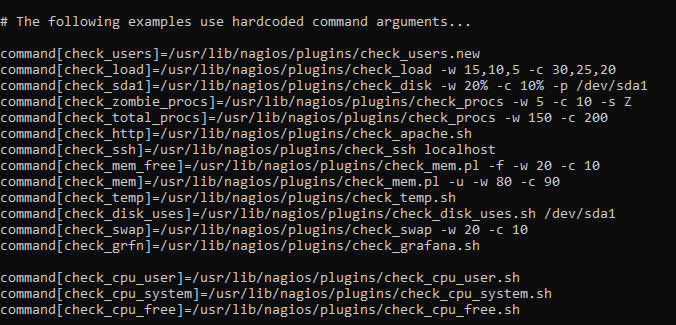
Antes de continuar conviene explicar una cosa. Desde el servidor Nagios, cuando definamos los servicios a monitorizar no podremos ajustar los argumentos según queramos, es decir, los limites para los warnings y los criticals, así como otros parámetros como la ruta a monitorizar para check\_disk etc. Estos ajustes deberán ser indicados en el fichero “nrpe.cfg” de cada equipo. Este fichero se encuentra en */etc/nagios/nrpe.cfg*, deberemos modificar las siguientes líneas para poder monitorizar correctamente el equipo.

La primera línea indica que host tiene permiso de conectar con el equipo y llamar a los plugins.  
 | allowed\_hosts=127.0.0.1, suso.\*\*\*\*\*\*\*\*\*.com

La siguiente línea nos permitirá incluir argumentos mediante variables que serán pasadas desde el servidor nagios a la hora de llamar al comando/script que nos devolverá las métricas.  
 | dont\_blame\_nrpe=1

### Personalizando los comandos NRPE

Casi al final del fichero “*nrpe.cfg”* encontramos los comandos que vienen definidos por defecto, podemos editarlos, eliminarlos o incluso crear nuevos comandos. En nuestro caso tenemos ya unos cuantos personalizados y creados.



Como podemos ver, el proceso para definir el comando es bastante sencillo.   
command[nombrecomando]=/path/al/script/perlshpythonentreotros.extension argumentos (si los necesita)  
Es importante indicar que el [nombrecomando] es el nombre que usará después el servidor de Nagios para llamar a ese comando, podemos poner el nombre que nos apetezca, pero ha de ser el mismo en el servidor de Nagios.

### Comprobaciones desde Nagios

Para comprobar que el host está correctamente configurado y que el servidor Nagios se comunica correctamente con el podemos ejecutar el siguiente comando en el servidor:  
 | $ /usr/lib/nagios/plugins/check\_nrpe -H IPDELHOSTODNS

En caso de estar todo correcto nos devolverá algo del tipo:   


Si queremos probar el funcionamiento pasando argumentos la sintaxis será, p.ej:  
| $ /usr/lib/nagios/plugins/check\_nrpe -H IPDELHOSTODNS -c comandodefinidoenelhost -a |$arg1 $arg2…



Con el parámetro -c le indicamos el comando que ha de ejecutar en el host  
Con el parámetro -a le indicamos cada uno de los argumentos que necesita que le pasemos al comando.

## Configurando Nagios

Aquí haremos una parada para comentar qué es un objeto en Nagios. Esto no es más que el nombre que reciben las definiciones de hosts, contactos, servicios, comandos etc… Por lo tanto de aquí en adelante utilizaremos este término para referirnos a ellos.

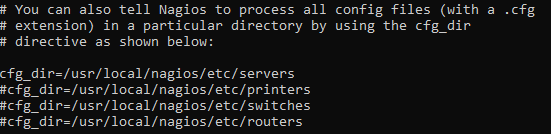
La definición de los objetos se realiza en ficheros con extensión “.cfg”. Estos ficheros pueden ser creados en cualquier parte, pero deben ser incluidos en */usr/local/nagios/etc/nagios.cfg*, y el usuario de Nagios debe tener permisos para leerlo.

Sin embargo, Nagios ya trae unos cuantos ficheros en la ruta */usr/local/nagios/etc/objects* para la definición de los objetos. Para los equipos Linux, en nuestro caso, el único fichero que hay es *“localhost.cfg”*. Si le echamos un vistazo observaremos que ya vienen definidos como ejemplo el host, un hostgroup y varios servicios (locales).

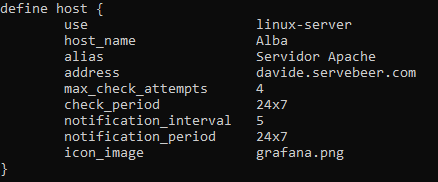
Una buena práctica antes de editar ningún fichero de configuración es hacerle una copia de seguridad. A la hora de guardar los ficheros que editemos y antes de reiniciar el servicio, deberemos comprobar que el usuario nagios tiene permisos de lectura (como mínimo) en esos ficheros.

### Definiendo el host

Como se ha comentado antes, podemos definir los hosts donde queramos, siempre que lo indiquemos en el fichero de configuración de Nagios. Con el fin de tener un mayor orden, crearemos un fichero para monitorizar equipos Linux y lo almacenaremos en */usr/local/nagios/etc/servers/*

En el fichero de configuración de Nagios le indicamos la ruta en la que almacenaremos los ficheros de cada host.  
 | # nano /usr/local/nagios/etc/nagios.cfg   
  
Encontraremos una sección del fichero en la que se nos permite incluir rutas en las que almacenar los ficheros .cfg, le indicamos la nuestra.   


Ahora, en la misma ruta que hemos indicado creamos un fichero, en el cual definiremos uno de los host y los servicios del host que se van a monitorizar. Para este ejemplo crearemos el fichero apache.cfg



Aclaraciones:

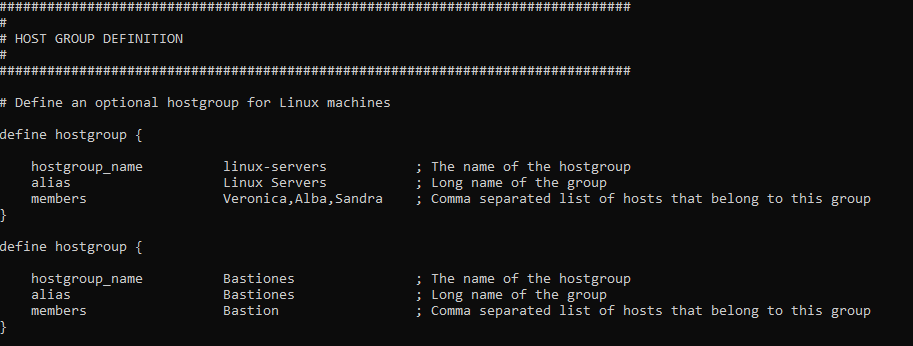
* Use: Con esta directiva se le indica una plantilla de la que heredar la configuración. En caso de conflicto porque una misma directiva se utilice tanto en la plantilla como en la definición del host, siempre tendrá prioridad el valor que se establece en la definición del host. De momento usaremos esta plantilla y más adelante las veremos más detalladamente.
* host\_name: Nombre corto usado para identificar el host.
* alias: Nombre o descripción usada para identificar al host
* address: Dirección IP del equipo a monitorizar (se pueden usar nombres DNS)
* max\_check\_attempts: Cantidad de veces que reintentará Nagios comprobar el servicio en caso de error
* check\_period: Horario en el que se comprobará el servicio, hay un fichero con las plantillas de los horarios en */usr/local/nagios/etc/objects/timeperiods.cfg*
* notification\_interval: Intervalo de tiempo entre notificaciones de caída de servicio, expresado en minutos.
* notification\_period: La pesadilla de todo IT, serás notificado 24x7 si el servicio se cae. También hay una definición en la plantilla timeperiods.cfg para workhours (8x5)
* icon\_image: Añade un icono junto al nombre del objeto. Este ha de ser 40x40 en formato png (preferentemente) o jpg.

Aquí se pueden establecer muchas mas directivas, y no todos los hosts tienen que estar definidos de la misma forma. Unos pueden tener unos valores y unas directivas y otros otras. Incluso se puede evitar la utilización de plantillas, aunque siempre se deben incluir algunas directivas que son obligatorias, ya sean puestas explícitamente o heredadas de una plantilla.

### Definiendo el hostgroup

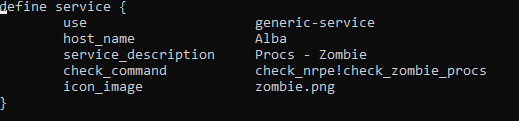
Los hostgroup no son obligatorios pero pueden ser muy útiles. Estos pueden ser usados para dos cosas:

* Agrupar equipos a la hora de visualizarlos en la interfaz web
* Facilitar la gestión. Por ejemplo, se puede definir un servicio que será aplicado a todos los equipos de un hostgroup.

En nuestro caso hemos definido los hostgroups y los equipos que lo componen en el fichero */usr/local/nagios/etc/objects/localhost.cfg.*

### Definiendo servicios en Linux con check\_nrpe

La definición del servicio la vamos a realizar en el fichero de cada host, aunque se podrían definir los servicios en un fichero independiente. Por ejemplo, el servicio de check\_zombie\_procs (procesos zombi) sería:

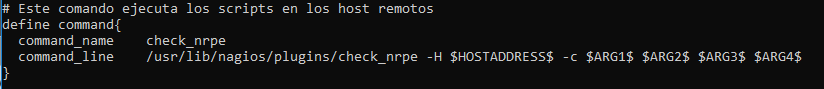


Aclaraciones:

* use: Como en el caso de la definición del host, esta directiva se emplea para utilizar na plantilla, en este caso *“generic-service”.*
* host\_name: Es el nombre del host o hosts a los que se le aplicará la monitorización de este servicio. Si quisiéramos especificar un hostgroup podríamos hacerlo utilizando la directiva *hostgroup\_name* en su lugar.
* service\_description: Nombre descriptivo para el servicio.
* check\_command: El comando que usará este servicio junto con su variable, en este caso *check\_zombie\_procs*

### Definiendo el comando

Por último, antes de reiniciar y ver que todo funciona bien tenemos que definir el comando check\_nrpe en el fichero */usr/local/nagios/etc/objects/commands.cfg*.



En caso de estar usando NRPE sin pasar argumentos desde Nagios no necesitaríamos añadir $ARG2$ en adelante.  
Una vez terminado esto ya podemos reiniciar Nagios y ver que la monitorización se realiza correctamente.

## Configurando y modificando las plantillas

Ya hemos visto como definir hosts, servicios y comandos, pero en estos utilizábamos plantillas y no configurábamos todas las directivas. En este punto vamos a editar el fichero *“templates.cfg”* que contiene plantillas ya definidas para contactos, hosts y servicios.

### Modificando plantillas para los hosts

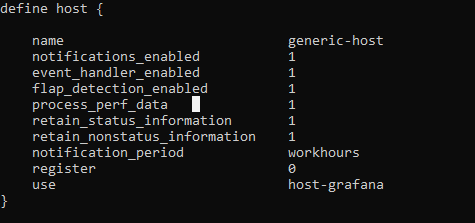
Si vamos al fichero *“templates.cfg”* veremos cinco plantillas predefinidas para los hosts

* generic-host: Plantilla genérica que usan el resto de plantillas
* linux-server: Plantilla para servidores Linux
* windows-server: Plantilla para servidores Windows.
* generic-printer: Plantilla para monitorizar impresoras.
* generic-switch: Plantilla para monitorizar switches.

#### Plantilla generic-host

Al ser la plantilla genérica, usada por el resto de plantillas, hay en ella ciertas directivas que conviene tener en cuenta para personalizar nuestro sistema de monitorización. La mayoría de sus directivas tiene valor “0” (desactivado) o “1” (activado).

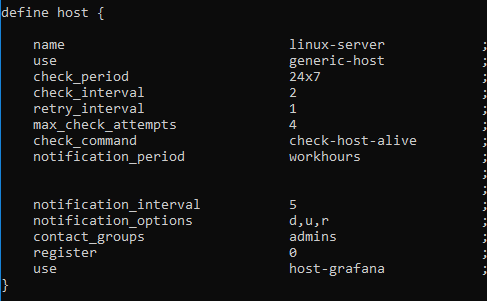
Aunque definamos en esta plantilla algunas directivas con un valor, luego en la plantilla del host donde la apliquemos podemos llamar a cualquier directiva de esta misma plantilla para asignarle un valor diferente si fuese necesario.



Algunas de las directivas que podemos encontrar en esta plantilla son:

* name: Nombre utilizado para la directiva
* notifications\_enabled: Si está activada enviará notificaciones al/los contacto/s definidos.
* event\_handler\_enabled: Habilita o no el control de eventos, útil para lanzar un evento en caso de alerta, p.ej: levantar un servicio tras un numero de comprobaciones con resultado critical.
* Flap\_detection\_enabled: Habilita o no la detección de “flapping”.
* Notification\_period: Especifica el nombre del “time period” en el que se permite el envío de notificaciones.

#### Plantilla linux-server



Aquí podemos observar que algunas directivas se repiten y sobrescriben:

* check\_period: Al igual que se puede indicar el time period en el que se pueden enviar notificaciones, también se puede establecer uno para indicar cuando se monitorizará.
* check\_interval: Indica cada cuanto tiempo se comprobará el objeto. Expresado en minutos.
* retry\_interval: Si un host cambia de estado hará los checks con más frecuencia hasta que cambie a un estado de tipo HARD. Expresado en minutos.
* max\_check\_attempts: Numero de checks que hará cuando el objeto cambie a un estado que no sea OK.
* check\_command: Comando utilizado para, en este caso, comprobar si el host está encendido.
* notification\_interval: Indica el tiempo en minutos que esperará hasta enviar una nueva notificación si el host aún sigue en un estado que no sea OK.
* notification\_options: Especifica que tipo de notificaciones se enviarán. Se pueden enviar notificaciones cuando el host pase a estos estados:
  + w: Warning
  + u: Unknown
  + c: Critical
  + f: Flapping
  + s: Scheduled downtime
  + r: Recovery
  + n: None

Las notificaciones de tipo recovery se producen cuando el host o servicio se ha recuperado de un estado anterior, es decir, pasa a un estado OK. Las de tipo “Scheduled downtime” se realizan cuando se ha programado un tiempo de apagado. Por ejemplo, apagamos un host para realizarle un mantenimiento y queremos o no, que se nos notifique cuando el tiempo de apagado empiece y termine.

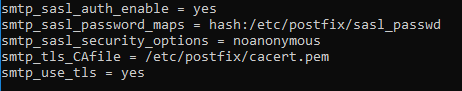
## Configurando las alertas vía correo

Una de las características de Nagios es la de poder enviar notificaciones a ciertas personas cuando ocurre algo. Así, si un equipo está apagado, tiene problemas de algún tipo, un servicio no funciona, etc, además de mostrarlo en la interfaz web, podrá enviar una notificación al personal oportuno.

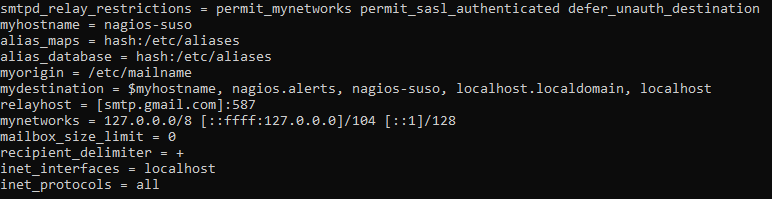
### Instalando y configurando el correo

Las alertas por correo pueden configurarse de muchas formas, en nuestro caso lo hemos hecho instalando los paquetes mailutils y postfix, que permitirán a nuestro servidor Nagios enviar correos al exterior. En nuestro caso, y para evitar que los correos acaben en la bandeja de spam, hemos creado una cuenta de correo en Gmail, la cual configuraremos como smtp en el servicio de postfix para enviar desde ahí los correos. Comenzamos por instalar los paquetes necesarios:   
 | # apt-get install postfix mailutils

Durante la instalación de postfix nos saldrá una ventana en la que podremos configurar el tipo y nombre de nuestro servidor de correo. En nuestro caso elegimos Internet Site y de nombre podemos usar el que más nos guste (nagios.alerts en nuestro caso). Al finalizar la instalación debemos de configurar postfix, para ello editamos el fichero “*/etc/postfix/main.cf”*. Al final del fichero añadiremos las siguientes líneas, sirven para indicar el certificado de seguridad (usaremos uno local) y las credenciales de la cuenta. Las líneas que ya se encuentren en el fichero se recomienda comentarlas.



En el mismo fichero deberemos editar/incluir las siguientes líneas.



Una vez tengamos el fichero editado deberemos generar el fichero de credenciales para Gmail, podemos usar nuestro editor de preferencia para crearlo. El nombre del fichero lo hemos definido en *“main.cf”*, por lo que deberá estar en la ruta y tener el nombre que le indicásemos anteriormente. En nuestro caso es *“/etc/postfix/sasl\_passwd”. Tan solo tendremos que añadir la siguiente línea:*

*| [smtp.gmail.com]:587* [*cuentadegmail@gmail.com:Contraseña*](mailto:cuentadegmail@gmail.com:Contraseña)

*Deberemos de asignarle permisos al fichero, cargar el fichero en el mapeo de archivos de postfix y establecer permisos al fichero que se nos creará como base de datos de ficheros. Para ello ejecutaremos los siguientes comandos:*

*| # chmod 600 /etc/postfix/sasl\_passwd  
 | # postmap /etc/postfix/sasl\_passwd  
 | # chmod 600 /etc/postfix/sasl\_passwd.db*

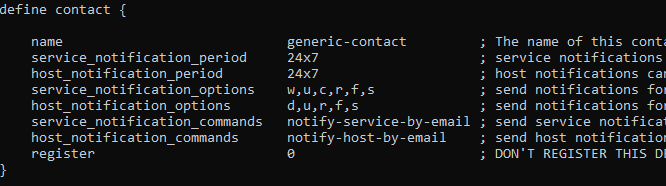
*Tras configurar todo esto debemos reiniciar el servicio y probar que podemos enviar correos de forma manual. En el siguiente punto detallamos el proceso de configuración en Nagios.*

*| # systemctl restart postfix  
 | # echo “Cuerpo del mensaje de prueba” | mailx -s Prueba”* [*destinatario@correo.com*](mailto:destinatario@correo.com)

*Si todo ha ido bien deberemos de recibir el correo. Si por el contrario, no recibimos nada, podemos revisar el log del servidor para ver donde está el error, dicho log se almacena en “/var/log/mail.info”*.

### Definiendo contactos en Nagios

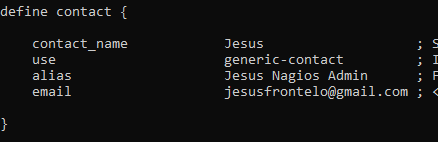
Ya tenemos configurado Nagios y el correo, ya solo quedan los contactos. En el fichero *“/usr/local/nagios/etc/objects/templates.cfg”* hay una plantilla para definir los contactos.



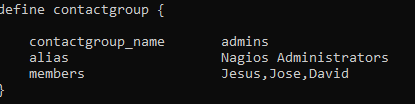
Aclaraciones:

* service\_notification\_period: Periodo de tiempo en el cual el contacto puede ser notificado acerca de problemas con los servicios.
* host\_notification\_period: Igual que el anterior pero para las notificaciones de los hosts.
* service\_notification\_options: Tipo de notificaciones que serán enviadas para los servicios.
* host\_notification\_options: Igual que el anterior pero con los estados de los hosts.
* service\_notification\_commands: Comando que se ejecuta al enviar una notificación sobre un servicio.
* host\_notification\_commands: Igual que el anterior pero para las notificaciones de los hosts.

Los contactos, al igual que otros objetos, también tienen su fichero particular. En este caso es *“contacts.cfg”*.



En el mismo fichero viene una definición para grupo de contactos.



### Definiendo el comando para enviar notificaciones

Una vez tengamos configurado el servidor de correo y los contactos, es hora de definir los comandos que se ejecutarán en caso de alerta. Por suerte estos comandos vienen predefinidos tanto para notificar alertas de host como para alertas de servicios. En nuestro caso solo hemos de modificar la parte señalada en negrita en cada comando.

# 'notify-host-by-email' command definition

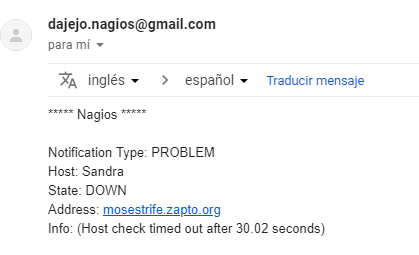
define command{  
command\_name notify-host-by-email  
command\_line /usr/bin/printf "%b" "\*\*\*\*\* Nagios \*\*\*\*\*\n\nNotification Type: $NOTIFICATIONTYPE$\nHost: $HOSTNAME$\nState: $HOSTSTATE$\nAddress: $HOSTADDRESS$\nInfo: $HOSTOUTPUT$\n\nDate/Time: $LONGDATETIME$\n" | **/usr/bin/mailx** -s "\*\* $NOTIFICATIONTYPE$ Host Alert: $HOSTNAME$ is $HOSTSTATE$ \*\*" $CONTACTEMAIL$  
}

# 'notify-service-by-email' command definition

define command{  
command\_name notify-service-by-email  
command\_line /usr/bin/printf "%b" "\*\*\*\*\* Nagios \*\*\*\*\*\n\nNotification Type: $NOTIFICATIONTYPE$\n\nService: $SERVICEDESC$\nHost: $HOSTALIAS$\nAddress: $HOSTADDRESS$\nState: $SERVICESTATE$\n\nDate/Time: $LONGDATETIME$\n\nAdditional Info:\n\n$SERVICEOUTPUT$\n" | **/usr/bin/mailx** -s "\*\* $NOTIFICATIONTYPE$ Service Alert: $HOSTALIAS$/$SERVICEDESC$ is $SERVICESTATE$ \*\*" $CONTACTEMAIL$  
}

### Comprobaciones

Una vez configurado el correo y los contactos, si nuestros hosts o servicios tienen un problema recibiremos una notificación en nuestro correo. Para probar esto basta con causar algún problema, por ejemplo, desconectamos el cable del servidor de base de datos (Sandra) y esperamos a que Nagios cambie su estado a **Down** de tipo **Hard.**



## Creando plugins de monitorización

Para crear un servicio propio debemos definir el comando y dirigirlo a un script. Nagios trae por defecto bastantes scripts propios en ficheros en código binario, además nos podemos descargar más de la página oficial de Nagios. Pero también podemos crear los nuestros.

Estos scripts van a estar escritos en Shell Script. Los vamos a crear en la misma carpeta en la que vienen por defecto, pero podemos crearlos en cualquier lugar en que el usuario nagios tenga permisos para llegar, además deberán tener permisos de ejecución (+x) para poder ejecutarlos.

Para empezar con el script debemos saber cómo sacar la métrica que queremos monitorizar. Como ejemplo vamos a comprobar la temperatura de la CPU. Esto se averigua con el siguiente comando:

| # cat /sys/class/thermal/thermal\_zone\*/temp

Esto nos devuelve la temperatura que buscamos multiplicada por 1000. De manera que guardaremos dicho valor en una variable y lo dividiremos por 1000:



Ahora vamos a definir la salida del comando, lo cual debe devolver el mensaje que escribirá Nagios. Creamos una variable que almacenará el texto, el valor y el performance data que se usará en todas las salidas (OK, WARNING, CRITICAL, UNKNOWN):

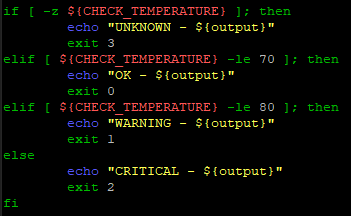


Aclaraciones:

La variable contiene un mensaje que nos devolverá el texto “CPU is <temperatura>ºC”, al otro lado del pipe “|” se establecen los valores del performance data:

* cpu\_temp: El nombre que recibe el dato.
* ${CHECK\_TEMPERATURE}: Es el valor que recibe el dato.
* ºC: Las unidades del dato, en este caso Grados Celsius.
* 70: Valor de Warning
* 80: Valor de Critical.

Nagios utiliza 4 tipos de salida, por lo que deberemos definir cuatro mensajes diferentes para cada uno de los tipos de salida.



Como podemos observar, los estados de nagios vienen definidos por el código de salida:

* exit 0: Si el valor entra en el umbral definido como ok nagios obtendrá el resultado como OK.
* exit 1: Si el valor obtenido entra en el umbral definido como warning, nagios obtendrá el resultado como WARNING.
* exit 2: Si el valor obtenido entra en el umbral definido como critical nagios obtendrá el resultado como CRITICAL
* exit 3: Si el valor obtenido es incorrecto, nulo o desconocido nagios obtiene el resultado como UNKNOWN.

La salida del código de error es importante, puesto que Nagios no comprueba el mensaje que le pasemos, sino que comprobará el código de error que devuelve. Por ello si es OK debe devolver 0, de no ser así podríamos encontrarnos con algo como esto:



## Creando Event-Handlers

Con Nagios podemos aplicar ciertos procedimientos para intentar recuperar automáticamente un servicio que está fallando. Los event-handler combinados con nrpe pueden ayudarnos a ello.

Mediante el event-handler podemos ejecutar un comando cada vez que un servicio cambia de estado. Para saber dónde estamos tenemos tres variables.

* $SERVICESTATE$: El estado del servicio puede ser OK, WARNING, UNKNOWN O CRITICAL.
* $SERVICESTATETYPE$: Puede ser HARD (Ha llegado al número máximo de intentos) o SOFT (aún no ha llegado al máximo de intentos, por lo que aún no se ha mandado la notificación)
* $SERVICEATTEMPT$: El número de veces que se ha intentado el check en el estado actual.

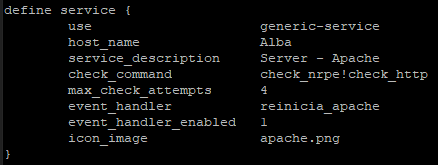
Por lo que en un check que tuviera configurados 3 intentos el orden sería:

* CRITICAL SOFT 1: Primera vez que falla.
* CRITICAL SOFT 2: Segunda vez consecutiva que falla.
* CRITICAL HARD 3: Tercera vez consecutiva que falla y momento en el que se mandan las notificaciones.

Para poder utilizar los event handler necesitamos activar la opción de manera global, esto se hace en el fichero de configuración de nagios *“/usr/local/nagios/nagios.cfg”.*

| enable\_event\_handlers=1

Ahora ya podemos activar en los servicios que deseemos los event handler e indicar cual será el evento que se ejecutará, normalmente un comando o un script. Como ejemplo vamos a activarlo en el servicio de apache, de manera que si este servicio se llegase a caer, al 4 intento de check ejecutaría el evento handler. Para ello editamos el servicio definido en el fichero *“/usr/local/nagios/etc/servers/apache.cfg”*.



En el ejemplo el event-handler que se va a ejecutar se llama reinicia\_apache, como ya explicamos anteriormente, tenemos que definir los comandos, para este ejemplo lo hemos realizado en el fichero *“/usr/local/nagios/etc/objects/commands.cfg”*.

| define command{  
| command name reinicia\_apache  
| command\_line $USER2$/restart\_apache.sh $SERVICESTATE$ $SERVICESTATETYPE$   
| $SERVICEATTEMPT$ $HOSTADDRESS$  
| }

En nuestro caso, el comando llama a un script que hemos creado en la ruta *“/usr/local/nagios/libexec/eventhandlers”, el cual va a recibir las variables anteriormente citadas y le añadimos otra más, que contiene la dirección (IP o DNS) del host que tiene el servicio caído. El script contiene lo siguiente:*



Como se puede observar, es una estructura condicional, en la que solo nos interesa cuando el servicio está en CRITICAL, en cuyo caso comprobará las variables $SERVICESTATE$ y $SERVICESTATETYPE$ para realizar las acciones. Como podemos ver, $4 hace referencia a $HOSTADDRESS$, necesaria para que el comando npre sepa a que host atacar.

Para terminar, solo hemos de definir los comandos a los que llamamos en cada estado del case, estos se definen en el host. 

# Instalación de InfluxDB y configuración con Nagios (nagflux).

## Requisitos.

La instalación de InfluxDB requiere de privilegios administrativos, por lo que el procedimiento se realizará con el usuario root.   
Por defecto, el servicio de InfluxDB necesita los siguientes puertos de red:

* El puerto TCP 8086 se utiliza para la comunicación cliente-servidor a través de la API HTTP de InfluxDB.
* El puerto TCP 8088 se utiliza para el servicio RPC para copia de seguridad y restauración. En nuestro caso no lo vamos a usar, pero sería aconsejable para no perder las métricas.

InfluxDB utiliza la hora local del host en UTC para asignar marcas de tiempo a los datos y para fines de coordinación. Deberemos de utilizar el protocolo de tiempo de red (NTP) para sincronizar el tiempo entre hosts. Si los relojes de los hosts no están sincronizados con NTP, las marcas de tiempo en los datos escritos en InfluxDB pueden ser inexactas.

La instalación de InfluxDB la realizaremos haciendo uso de los repositorios oficiales de InfluxData, empresa desarrolladora de InfluxDB. Para ello incluimos la URL del repositorio a nuestro fichero de repositorios.  
| # curl -sL <https://repos.influxdata.com/influxdb.key> | sudo apt-key add – source |/etc/osrelease  
| # test $VERSION\_ID = "9" && echo "deb https://repos.influxdata.com/debian stretch |stable" | sudo tee /etc/apt/sources.list.d/influxdb.list

## Instalando InfluxDB

El proceso de instalación de InfluxDB es bastante sencillo. Tan solo instalar el paquete e iniciar el servicio.  
 | # apt-get update && apt-get install influxdb  
 | # systemctl start influxdb

### Configuración de InfluxDB

El Sistema tiene valores predeterminados internos para cada configuración de archivo de configuración. Se pueden ver estos ajustes predeterminados con el comando.  
 | $ influxd config

El fichero de configuración local *“/etc/influxdb/influxdb.conf”* tiene la mayoría de las configuraciones comentadas. Si deseamos modificar o habilitar alguna configuración, será ahí donde lo tengamos que hacer.  
Para habilitar el fichero de configuración, tanto si hemos modificado el predeterminado, como si hemos creado uno nuevo, el comando será:  
 | # influxd -config /etc/influxdb/influxdb.conf (o la ruta al fichero creado en su defecto)  
Para los ficheros de configuración personalizados hemos de asignar los siguientes permisos.  
 | # chown influxdb:influxdb /ruta/al/fichero (tanto al fichero como a la ruta)

## Instalación y configuración de Nagflux.

Nagflux es un conector que transforma los datos de rendimiento de Nagios a InfluxDB. Recopila datos de los *Spoolfiles de Nagios* y agrega información de Livestatus. Estos datos se envían a InfluxDB , y serán consultados por Grafana cuando empecemos a crear las gráficas.

LiveStatus

Spoolfiles

InfluxDB

### Requisitos.

Al igual que InfluxDB, Nagflux está programado en lenguaje Google, por lo que necesitamos instalar los paquetes y las dependencias necesarias previas a la instalación.  
 | # apt-get install -y golang golang-github-influxdb-usage-client-dev git

### Instalando Nagflux.

Una vez cumplido el requisito anterior procedemos a la instalación del conector.  
Para instalarlo primero hemos descargar y compilar Nagflux en lenguaje Google.

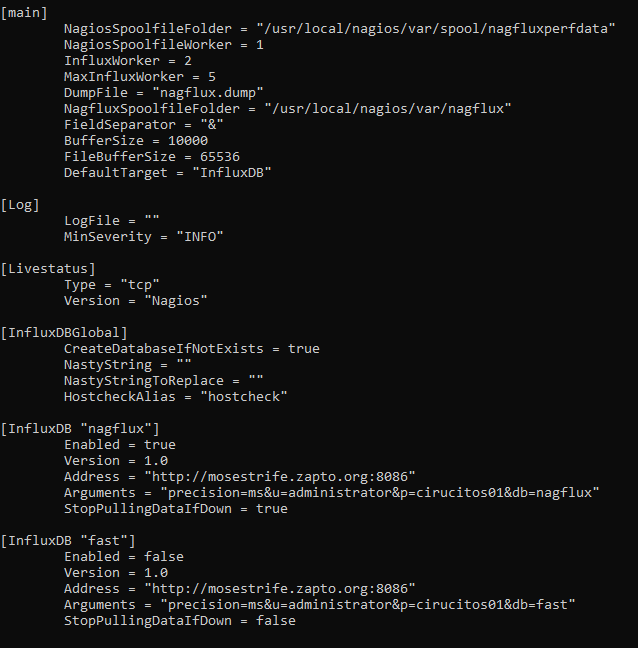
| # export GOPATH=$HOME/gorepo  
 | # mkdir $GOPATH  
 | # go get -v -u github.com/griesbacher/nagflux  
 | # go build github.com/griesbacher/nagflux  
 | # mkdir -p /opt/nagflux  
 | # cp $GOPATH/bin/nagflux /opt/nagflux  
 | # mkdir -p /usr/local/nagios/var/spool/nagfluxperfdata  
 | # chown nagios:nagios /usr/local/nagios/var/spool/nagfluxperfdata

El siguiente paso será crear el servicio y habilitarlo para que se cargue al inicio del sistema.

| # cp $GOPATH/src/github.com/griesbacher/nagflux/nagflux.service /lib/systemd/system  
 | # chmod +x /lib/systemd/system/nagflux.service  
 | # systemctl daemon-reload  
 | # systemctl enable nagflux.service

### Configurando Nagflux

Una vez instalado el conector que se encargará de transmitir el performance data de Nagios hacia la base de datos InfluxDB, hemos de configurar como va a recolectar ese performance data, como lo va a transformar para introducirlo en la base de datos y como se va a conectar con la base de datos. Lo primero que hemos de hacer es crear el fichero de configuración, este ha de contener la siguiente información.



contraseña

Usuario

URL InfluxDB

URL InfluxDB

contraseña

Usuario

Para hacerlo mas simple, podemos hacer uso del siguiente *“script”*, donde solo hemos de modificar la información que necesitemos según nuestra infraestructura.

cd /opt/nagflux  
printf '[main]\n' > config.gcfg  
printf '\tNagiosSpoolfileFolder = "/usr/local/nagios/var/spool/nagfluxperfdata"\n' >> config.gcfg  
printf '\tNagiosSpoolfileWorker = 1\n' >> config.gcfg  
printf '\tInfluxWorker = 2\n' >> config.gcfg  
printf '\tMaxInfluxWorker = 5\n' >> config.gcfg  
printf '\tDumpFile = "nagflux.dump"\n' >> config.gcfg  
printf '\tNagfluxSpoolfileFolder = "/usr/local/nagios/var/nagflux"\n' >> config.gcfg  
printf '\tFieldSeparator = "&"\n' >> config.gcfg  
printf '\tBufferSize = 10000\n' >> config.gcfg  
printf '\tFileBufferSize = 65536\n' >> config.gcfg  
printf '\tDefaultTarget = "all"\n' >> config.gcfg  
printf '\n' >> config.gcfg  
printf '[Log]\n' >> config.gcfg  
printf '\tLogFile = ""\n' >> config.gcfg  
printf '\tMinSeverity = "INFO"\n' >> config.gcfg  
printf '\n' >> config.gcfg  
printf '[InfluxDBGlobal]\n' >> config.gcfg  
printf '\tCreateDatabaseIfNotExists = true\n' >> config.gcfg  
printf '\tNastyString = ""\n' >> config.gcfg  
printf '\tNastyStringToReplace = ""\n' >> config.gcfg  
printf '\tHostcheckAlias = "hostcheck"\n' >> config.gcfg  
printf '\n' >> config.gcfg  
printf '[InfluxDB "nagflux"]\n' >> config.gcfg  
printf '\tEnabled = true\n' >> config.gcfg  
printf '\tVersion = 1.0\n' >> config.gcfg  
printf '\tAddress = "http://127.0.0.1:8086"\n' >> config.gcfg  
printf '\tArguments = "precision=ms&u=root&p=root&db=nagflux"\n' >> config.gcfg  
printf '\tStopPullingDataIfDown = true\n' >> config.gcfg  
printf '\n' >> config.gcfg  
printf '[InfluxDB "fast"]\n' >> config.gcfg  
printf '\tEnabled = false\n' >> config.gcfg  
printf '\tVersion = 1.0\n' >> config.gcfg  
printf '\tAddress = "http://127.0.0.1:8086"\n' >> config.gcfg  
printf '\tArguments = "precision=ms&u=root&p=root&db=fast"\n' >> config.gcfg  
printf '\tStopPullingDataIfDown = false\n' >> config.gcfg

Una vez creado el fichero tan solo nos queda inciar el servicio para que cargue la configuracion.  
 | # systemctl start nagflux.service

### Comprobaciones

Cuando Nagflux se inicia por primera vez creará una base de datos en InfluxDB, en nuestro caso hemos hecho que esa base de datos se llame nagflux (al crear el fichero de configuración). Si todo está correcto, podemos comprobar que se ha creado ejecutando el siguiente comando:

| # curl -G "http://localhost:8086/query?pretty=true" --data-urlencode "q=show databases"

Nos ha de devolver una salida como la siguiente, donde apreciaremos el nombre de la base de datos *“nagflux”.*

{  
 "results": [  
 {  
 "statement\_id": 0,  
 "series": [  
 {  
 "name": "databases",  
 "columns": [  
 "name"  
 ],  
 "values": [  
 [  
 "\_internal"  
 ],  
 [  
 "nagflux"  
 ]  
 ]  
 }  
 ]  
 }  
 ]  
}

Si el nombre de la base datos aparece en la lista de bases de datos quiere decir que la configuración es correcta e InfluxDB está preparado para recibir el performance data desde Nagios.

## Configuración y comandos de Nagios para enviar PerfData a InfluxDB.