PushMon. Configuración y análisis.

Introducción.

Echando un ojo a Nagios Exchange me encontré con Pushmon este pequeña y parece reciente joya que no conocía. Se trata de un agente para Nagios con enfoque "Push", o sea, es el propio cliente el que realiza los chequeos y se los manda al servidor de Nagios mediante un intermediario en este. Todos los componentes están desarrollados en Python con un enfoque que puede tener mucho recorrido. Es ideal para servidores remotos con difícil comunicación por configuraciones de seguridad y para servidores en la Nube ya que son los clientes los que comunican los resultados de los chequeos al servidor conectándose a una URI y usando REST. Probándolo encontré una "pequeña" pega y es que la parte servidora (al menos el servicio statusd) no funcionará tal cual en Centos/Redhat 6.x debido a la versión de Python de estos (2.6.x). Usa al menos un método (check_output) que es de Python 2.7. Aunque hay formas de instalar alternativamente Python 2.7 en estos sistemas... personalmente no me pondría a hacerlo en mis servidores. En Debian 7 no hay problema y como agente en cliente para CentOS / Redhat 6.x también funciona sin problemas.

Elementos / Servicios.

Veamos un poco en detalle los componentes que lo conforman.

En la parte **cliente** tenemos el "**checker daemon**". Este se ocupa de ejecutar los plugins de Nagios que le configuremos en el cliente y mandar los datos al servidor (statusd webapp). Los tiempos de chequeo para cada Plugin y de envió global de datos al servidor son configurables. Necesitaremos tener instalados los plugins de Nagios o cualquier otro plugin que vayamos a usar en local. Probado en Debian 7 y CentOS 6.4.

En el **servidor** tenemos los siguiente componentes:

- **BBDD Redis**: En esta bbdd se guardan datos de hosts, chequeos, usuarios,... de los que se sirve la aplicación para gestionar y generar ficheros de configuración para Nagios. Debe estar siempre iniciado el servicio de redis.
- **Statusd webapp:** Es una aplicación que filtra los envíos POST y salva los resultados para cada host en una BBDD redis. Deberá estar siempre iniciado. Es el único componente servidor que necesita tener una ip para acceso externo. El resto pueden atender solo el localhost si instalamos todos los servicios en el servidor de Nagios.
- **Resultsd webapp**: Esta aplicación es una api web que conecta con la bbdd redis y proporciona los datos para los hosts y chequeos. Tanto el ConfigBuilder como los chequeos de nagios dependen de este componentes para obtener los datos que necesitan. Estará siempre iniciado también logicamente.
- **Config-Builder**: Es un script también python que obtiene los resultados de los hosts (configuración de estos) conectando a la API de Resultsd y los usa para generar los ficheros de configuración de Nagios. Añade automáticamente tanto todos los chequeos configurados en cliente (recibidos a través de Statusd y alojados por este en la bbdd redis) como otros chequeos adicionales asociados. Se ejecuta solo cuando se añaden clientes o chequeos a estos, para que genere la nueva configuración de Nagios.
  Además de añadir los chequeos configurados en cliente añadirá otro adicional para verificar que el cliente esté mandando resultados en un tiempo prefijado y alertar si no es así.

- **Nagios Plugins:** Adicionalmente disponemos de plugins para verificar que los demonios Statusd y Results estén funcionando. Lo normal es que se los configuremos asociados al propio host de Nagios.

Instalación inicial. Servidor.

Instalamos primero los requisitos y algún paquete más que puede ser necesario. Se indica como instalarlo en CentOS pero como se comentaba no funcionará por el problema de Python. Queda escrito por si lo solucionaran.

Para Debian y probablemente Ubuntu

```
apt-get install python-pip redis-server unzip
```

Para CentOS y probablemente Redhat

```
yum install python-pip redis unzip sudo
```

Verificar que el servicio de redis esté levantado!!! En Debian lo levanta al instarlo pero en CentOS no lo levanta. Debemos levantarlo y configurarlo para que lo haga al iniciar.

Podemos bajar tal como nos indican el sofware mediante git. En nuestro caso optamos por bajar directamente el zip y trabajar con él. Copiamos el zip en la carpeta que queramos tener el sofware. En nuestro caso usaremos /usr/local (en los ejemplos del site usan un home de usuario). Lo descomprimimos e instalamos los requisitos de módulos python mediante pip:

```
unzip PushMon-master.zip
cd PushMon-master
pip install -r requirements/all.txt
```

Configuración e inicio de statusd webapp

Estableceremos la siguiente variables para todo lo relativo a la configuración, de esta forma simplificamos el trabajo y los ejemplos nos valdrán para los directorios donde tengamos instalado todo:

Directorio de instalación del sofware PUSHMON, directorio de Plugins de Nagios y directorio de configuración de Nagios.

```
export PUSHMON=/usr/local/PushMon-master
export NAGIOSPLUGINS=/usr/lib/nagios/plugins
export NAGIOSCONFDIR=/etc/nagios3/conf.d
```

Configuración de parámetros.

Editamos el fichero de config. de statusd y cambiaremos la IP por la de nuestro servidor (externa) y añadiremos o no redes permitidas.

vim $PUSHMON/statusd/settings.yaml

```
######################################################################
# The server side settings file                                      #
######################################################################
```

```
host: 192.168.164.132
port: 4444
ssl:
  ssl_key: ssl/server.key
  ssl_cert: ssl/server.crt

allowed:
  - 10.0.0.0/8
  - 192.168.164.0/24
  - 192.168.1.0/24
  - 127.0.0.0/8

post_elements:
  - status
  - execution_time
  - last_run
  - perfdata
  - output
  - exitcode

handle_host_data: True
host_data_command: bin/get_host_data

redishost: 127.0.0.1
redisport: 6379
redisdb: 0
redispass: None
```

Generamos las claves.

```
Cd $PUSHMON/statusd/ssl/
./genkey.sh
```

Ejecutamos el servidor

Inicialmente lo ejecutamos en consola para ver que funcione. Luego lo ejecutaremos en segundo plano y sin el verbose. Estaría bien también currarse unos scripts de inicio que no existen :-)

```
cd $PUSHMON/statusd
./start-statusd.py -v -s  settings.yaml
```

Veremos con netstat que se está ejecutando el el puerto indicado (OJO, debemos también permitirlo en nuestro FW):

```
cp        0      0 192.168.164.132:4444    0.0.0.0:*               LISTEN
```

Tanto con este servicio, statusd, como con el servicio resultsd podemos asignarle a los directorios un usuario con permisos limitados y ejecutarlos con dicho usuario si queremos.

Configuración e inicio de resultsd

En este fichero de configuración vemos que hay unos usuarios predefinidos (en la BBDD de redis probablemente). De momento nos valen para hacer las pruebas. Usaremos localhost para la ejecución del servicio ya que no necesita contacto con el exterior.

```
cd $PUSHMON/resultsd
vim settings.yaml
```

```
######################################################################
# The server side settings file                      #
######################################################################

host: 127.0.0.1
port: 5555
ssl:
  ssl_key: ssl/server.key
  ssl_cert: ssl/server.crt

allowed:
  - 10.0.0.0/8
  - 192.168.164.0/24
  - 192.168.1.0/24
  - 127.0.0.0/8

admins:
  - admin
  - badong
  - anton_gleuf
  - flip.fliphess.com
  - flip

redishost: 127.0.0.1
redisport: 6379
redisdb: 0
```

Generamos las claves.

```
cd $PUSHMON/resultsd/ssl/
./genkey.sh
```

Ejecutamos el servidor

Lo ejecutamos para probar que no dé errores y salimos.

```
cd $PUSHMON/resultsd
./results-daemon.py -v -s settings.yaml
```

Configuración e inicio de config-builder

```
cd $PUSHMON/plugins/config-builder
```

Editamos el fichero settings.yaml y cambiamos los valores que nos afectan. Los usuarios los dejamos tal cual de momento. Ojo con los directorios para Nagios. Debemos crear un nuevo directorio y añadírselo en nagios.cfg o bien indicarle en config_dir un directorio existente ya de ficheros de configuración de Nagios. Hay será donde cree los objetos hosts/servicios de Push.

```
username: anton_gleuf
password: videopost
server_url: https://127.0.0.1:5555
verify_ssl: False
timeout: 5
config_dir: /etc/nagios3/conf.d/generated
tmpdir: /tmp/nagios_config
```

No iniciamos el demonio aún ya que nos faltan cosas por configurar:

Configuración de Sudoer

Necesitamos añadir la siguiente línea a sodoers para permitir ejecutar al usuario nagios estos dos comandos:

```
nagios  ALL=(ALL) NOPASSWD:/etc/init.d/nagios3, /usr/sbin/nagios3
```

Podemos seguir el procedimiento que nos facilitan para hacerlo

```
cp $PUSHMON/plugins/config-builder/files/checkrunner.sudofile /etc/sudoers.d/checkrunner
chmod 600 /etc/sudoers.d/checkrunner ; chown root:root /etc/sudoers.d/checkrunner
```

Ojo con las rutas de los comandos. Adecuarlas a las nuestras.

Configuración de plugins activos.

Necesitamos copiar y configurar los plugins activos de la parte servidor

```
cp $PUSHMON/plugins/nagios-checks/* $NAGIOSPLUGINS -R
cp $PUSHMON/plugins/config-builder/files/static-nagios-config.cfg  $NAGIOSCONFDIR/static.cfg
```

Editamos el archivo $NAGIOSCONFDIR/static.cfg y adecuamos las rutas. En nuestro caso como hemos copiado los plugins al directorio de plugins de Nagios quedaría de la siguiente forma.

```
#
# Static nagios configuration
#


# 'check-host-alive' command definition
#define command{
#       command_name  check-host-alive
#       command_line    /usr/lib/nagios/plugins/check_ping -H '$HOSTNAME$' -w 5000,100% -c 5000,100% -p 1
#       }
```

```
# the multiple commands checker
define command{
        command_name  get-results-for-check
        command_line    /usr/lib/nagios/plugins/check_service_for_client.py -H "$HOSTNAME$" -s
/usr/local/PushMon-master/plugins/nagios-checks/settings.yaml -C $ARG1$
        }

# check if node alive
define command{
        command_name  check-client-alive
        command_line    /usr/lib/nagios/plugins/check_client_alive.py -H "$HOSTNAME$" -w 100 -c 200 -s
/usr/local/PushMon-master/plugins/nagios-checks/settings.yaml
        }

# check if server alive
define command{
    command_name    check-server-alive
    command_line    /usr/lib/nagios/plugins/check_server_alive.py -s /usr/local/PushMon-master/plugins/nagios-
checks/settings.yaml
    }

# check all services for a client
define command{
        command_name  check-all-services-for-client
    command_line    /usr/lib/nagios/plugins/check_all_services_for_client.py -H "$HOSTNAME$" -s
/usr/local/PushMon-master/plugins/nagios-checks/settings.yaml
        }
```

Inicialmente copié y modifiqué el fichero settings.yaml al directorio de plugins de Nagios pero luego lo redirigí mejor a su ubicación original ya que statusd parece que lo lee en su ubicación original para ver la dirección del servidor redis al que conectarse. Mejor no tenerlo en dos sitios, aunque podriamos usar un enlace.

Modificamos el fichero $PUSHMON/plugins/nagios-checks/settings.yaml para reflejar todos los parámetros que hemos ido configurando.

```
statusd:
    username: admin
    password: badong
    verify_ssl: False
    timeout: 3
    server_url: https://192.168.164.132:4444/status

resultsd:
    username: anton_gleuf
    password: videopost
```

```
    verify_ssl: False
    timeout: 3
    server_url: https://127.0.0.1:5555/results
redis:
    redishost: 127.0.0.1
    redisport: 6379
    redisdb: 0
    redispass: None
```

Ejecutamos el Config Builder

cd $PUSHMON/plugins/config-builder
./dynamic-config-builder.py -s settings.yaml

Chequeará nuestra configuración de Nagios. Es posible que sea necesario cambiar algo en la configuración porque detecte problemas a raíz de los cambios introducidos.

Posibles problemas que nos podemos encontrar con Config Builder:

- Configuraciones de Nagios duplicadas. En mi caso dio un error debido a una configuración duplicada: Warning: Duplicate definition found for command 'check-host-alive' (config file '/etc/nagios3/conf.d/static.cfg'. Se comenta ese chequeo que ya existe la definición en Nagios.
- Si nos aparece un error del tipo “ValueError ("No JSON object could be decoded")” es que no puede contactar con la BBDD Redis. Seguramente tenemos el Servicio desactivado.
- Reinicio de Nagios y verificación: El script python tiene comandos en el propio script (los que añadió en sudoers). Los comandos son “sudo /usr/sbin/nagios3 -v /etc/nagios3/nagios.cfg” y “/etc/init.d/nagios3 reload”. Si tenemos una instalación sobre Debian nos puede coincidir pero si es sobre Redhat o editamos el script y lo cambiamos o hacemos enlaces simbólicos.
- Redhat / CentOS 6.x (NO USAR) Error similar a: “File "/home/push/PushMon-master/plugins/config-builder/lib/confighelpers.py", line 20, in generate_host_config data['host_data']['server_flags']['status'], \. KeyError: 'host_data'” Si examinamos el log de statusd veremos que nos aparece un error “Failed saving data for hostxxx! Error: 'module' object has no attribute 'check_output'.” El método check_output() method se define en python 2.7, si tenemos versión inferior (como Redhat /CentOS 6.4 esta incluye Python 2.6.6). Probablemte esto hace que introduzca errores en la bbdd y no lo registre correctamente.

http://stackoverflow.com/questions/4814970/subprocess-check-output-doesnt-seem-to-exist-python-2-6-5

http://www.bogotobogo.com/python/python_subprocess_module.php

Instalación. Parte cliente.

El cliente si nos funcionará tanto en Debian 7 como en CentOS/Redhat 6.4 (posiblemente 6.x)

Si vamos a realizar las pruebas de cliente en el propio servidor de Nagios ya lo tendremos instalado obviamente y podemos pasar a editar el archivo para configurarlo.

```
apt-get install python-pip redis-server unzip
```

Para CentOS / Redhat

```
yum install python-pip redis unzip
```

Copiamos el zip en /usr/local, lo descomprimimos e instalamos requisitos.

```
unzip PushMon-master.zip
cd PushMon-master
pip install -r requirements/client.txt
```

Editamos el archivo de configuración

Tenemos que editar el archivo para decirle donde está nuestro servidor u el username que es como creará nuestro nombre de host en Nagios. Password se deja tal cual.

```
cd $PUSHMON/client
vim settings.yaml
```

server: https://192.168.164.132:4444/status
username: server01.eldespistado.com
push_interval: 60
Todos los interval los ponemos a 60 (segs) para que no nos sature las pruebas tanto...
ATENCION a las rutas de los Plugins. Son correctas para Debian pero no para Otros Linux.
Evidentemente el único chequeo real (apt) solo será válido para Debian/ Ubuntu. Lo comentamos en CentOs.

El resto de líneas de chequeos son casi todas específicas para realizar pruebas. Debemos tener en cuenta que tenemos que tener los plugins standard de Nagios instalados en el cliente y que si no están en la ruta que indica el fichero debemos cambiarla.

Ejecutamos el cliente como siempre primero en consola para probarlo y luego ya en background.

```
./checkrunner-daemon.py -v -s settings.yaml
```

El cliente empezará a mandar datos al servidor y este a almacenarlos. Para que se genere la configuración de Nagios deberemos ejecutar el ConfigBuilder y posteriormente realizarle un restart / reload a Nagios para que añada el host generado en los ficheros de configuración por ConfigBuilder.

Configurando los plugins para controlar los servicios.

Necesitamos ahora configurar los plugins que monitorizan que nuestros servicios statusd y resultsd estén activos ya que si fallan, fallará la monitorización de todos los agentes que estén configurados con Pushmon.

Utilidades para manejar la BBDD Redis

En el directorio /usr/local/PushMon-master/plugins/tools incluye varias utilidades para “reiniciar” la BBDD, Borrar un Host, … Es conveniente después de hacer las pruebas iniciales reiniciar la BBDD.

PROBÁNDOLO.

Probando la configuración de ejemplo. Si tenemos levantado algún servicio lo matamos con un kill previamente. Es hora de probarlo. Levantamos en el servidor todos los servicios en background y logeando (de momento con verbose para obtener más datos en los logs en caso de problemas):

```
export PUSHMON=/usr/local/PushMon-master
cd $PUSHMON/statusd
./start-statusd.py -v -s  settings.yaml > statusd.log 2>&1 &
cd $PUSHMON/resultsd
./results-daemon.py -v -s settings.yaml > resultsd.log 2>&1  &
```

En el cliente levantamos y echamos un ojo a ver que hace:

```
export PUSHMON=/usr/local/PushMon-master
cd $PUSHMON/client
./checkrunner-daemon.py -v -s settings.yaml > client.log 2>&1 &
tail -f $PUSHMON/client/client.log
```

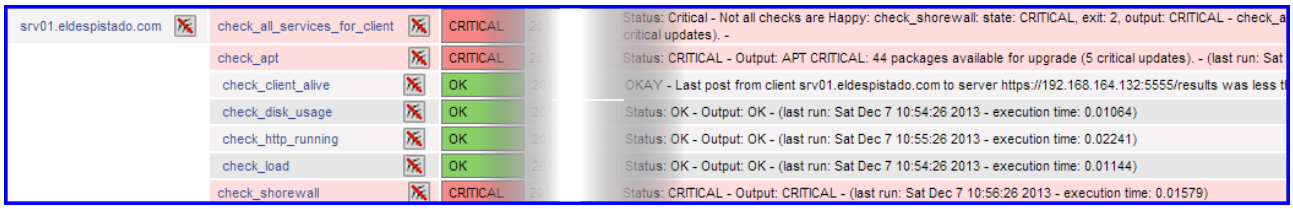
Podemos echar un ojo también al log de los servicios de servidor por si encontramos errores. Es en dicha salida donde veremos de forma bastante clara los posibles problemas.

Al rato (esperamos un minuto para darle tiempo) ejecutamos el Config Builder y vemos si ha generado ya el fichero de nuestro host en la ruta indicada previamente del directorio de configuración de Nagios.

```
cd $PUSHMON/plugins/config-builder
./dynamic-config-builder.py -s settings.yaml
```

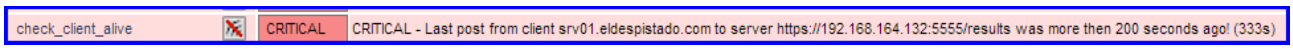
En nuestro caso la ruta es “/etc/nagios3/conf.d/generated/hosts”. Una vez generado podemos hacer un reload al Nagios para que creé el Host.

Nos encontramos a nuestro Host ya funcionando:



Tenemos los errores simulados por los chequeos de ejemplo con el Plugin dummy. Adicionalmente a los chequeos configurados en cliente tenemos otros adicionales generados en el servidor. Uno nos dá un sumario de todos los pugins y el otro y más importante “check_client_alive” nos avisa si el

host cliente no está mandando resultados en el tiempo prefijado. Si paramos el servicio en el cliente al rato veremos que nos alerta precisamente del problema:



Temas pendientes.

Quedarían varios temas pendientes de probar antes de poner en producción esta aplicación:

- Seguridad: Configurar la seguridad y realizar pruebas para que estos servicio se ejecuten con el mínimo nivel de privilegios. Tanto statusd como results los he probado con un usuario son privilegios y parece que no hay problema.

- Usuarios: cambiar los usuarios por defecto que usan los servicios para comunicarse entre ellos. Deben estar definidos en la bbdd Redis.

- Scripts de inicio. Habría que crear unos scripts decentes para iniciar, parar, verificar los servicios.

Conclusiones.

Sin duda un cliente muy interesante y prometedor... si continua su desarrollo. Existen varias formas (utilidades, clientes,...) en Nagios de realizar algo similar pero esta nos aporta una manera limpia y ordenada de hacerlo y sin necesidad de realizar la configuración de los chequeos en ambos lados (cliente y servidor).

El hecho de estar realizado en Phyton para mi tiene un valor añadido ya que pienso es un lenguaje de futuro para integrar componentes en Nagios y hacia el que parecen ir muchos desarrolladores que aportan a la comunidad Nagios.

En cualquier caso, aunque funcional, parece que el sofware aún está un tanto verde. Por las pruebas realizadas un fallo puede introducir errores en la BBDD Redis difíciles de diagnosticar. Los scripts de inicio no debería ser un problema ya que se pueden realizar sin mucho trabajo pero estaría bien que los incorporara.