UNIVERSIDAD POLITÉCNICA DE PUEBLA

INGENIERÍA EN INFORMÁTICA

PROYECTO DE ESTADÍA PROFESIONAL

“MONITOREO DE SERVIDORES CON NAGIOS Y SELENIUM”

MARTÍNEZ LÓPEZ RIGOBERTO

ASESOR TÉCNICO

JUAREZ LERMA IVÁN

ASESOR ACADEMICO

M.C RODRÍGUEZ HUESCA REBECA

Índice

Índice de figuras

Introducción

Las actividades diarias de alguna u otra forma se ven involucradas con servicios en línea, ya sea consultando nuestras actividades programadas del día, nuestras redes sociales o bien cuestiones de nuestro trabajo. Los servicios en la nube dependen de hardware físico que como cualquier máquina sufre desperfectos ya sea por mala operación o por errores de fábrica.

Siendo que las empresas dedicadas a este tipo de giro, sus dividendos dependen del estado de sus servicios ofertados es de gran importancia dar atención a problemas presentes en los equipos.

En este documento se presenta la implantación de un monitoreo con Nagios y el plugin NRPE, en el capítulo uno se habla sobre aspectos generales de la empresa y el porqué de la importancia del monitoreo implantada. En el segundo capítulo se desarrolla la metodología a seguir para el implantamiento del monitoreo.

Capítulo I

Grupo JOBO

1.1 Antecedentes

Grupo JOBO: nombre comercial de la empresa o BECITHEW SA DE CV: nombre registrado de manera oficial.
Grupo Jobo nació en el año 2006 de la mano de sus fundadores, dos jóvenes y exitosos empresarios quienes tuvieron la visión de realizar grandes proyectos de concepto innovador junto con un gran equipo de trabajo, el primer sitio lanzado fue Jobomas.com en febrero de 2008, desde aquél entonces la empresa se ha enfrentado a grandes desafíos, y gracias a ello surgieron nuevos sitios, nuevos proyectos y nuevas oportunidades que nos han hecho crecer como empresa a nivel mundial.

1.2 Sector y servicios

Empresa de desarrollo y administración de sitios web del sector de empleos.

1.3 Misión

Ser una empresa de desarrollo y administración de sitios web comprometida con la satisfacción de nuestros clientes y asociados, ofreciéndoles servicios de calidad con un alto nivel de competitividad y posicionamiento en el mercado.

1.4 Visión

Ser una empresa reconocida a nivel mundial como el conglomerado más grande de sitios web, abarcando diferentes temáticas y brindando herramientas tanto a los usuarios como a las empresas para facilitar su vida online.

1.5 Funciones del departamento donde se realizó el proyecto

Departamento de sistemas de Grupo JOBO es el encargado de administrar y monitorear los servidores de todos los proyectos con los que cuenta la empresa. Además de dar solución a los problemas recurrentes de los mismos en tiempo forma ya que la detención de algún sitio representa pérdidas económicas importantes para la empresa.

1.6 Descripción del proyecto

Grupo JOBO cuenta con los departamentos de Recursos Humanos, ejecutivos, programadores; el cual está divido por grupos de trabajos por proyectos, finanzas y departamento de sistemas, siendo éste último el encargado de llevar la administración y monitoreo de los servidores donde se alojan los diferentes servicios para poder mantener los variados sitios de la empresa funcionando de manera adecuada y en el cual el autor de este documento realizó su estadía profesional.

Dicha empresa tiene ocho sitios webs, el principal y más importante por la magnitud y trabajo detrás de este proyecto es jobomas.com, lanzado en 2008 con presencia en más de 60 países, algunos pertenecientes a LATAM, registrando en 2015 20, 000,000 de usuarios y 80,000 empresas ofertando empleos además de 30, 000, 000 de visitas mensuales. El proyecto que este documento describe se basa en el monitoreo de los recursos y servicios de los equipos donde se encuentran los sitios. Como herramienta principal de monitoreo se usó Nagios, una herramienta de código libre lanzada en 1999 siendo esta una de las herramientas número uno en el monitoreo de equipos de red y cómputo. Del lado de los hosts clientes o servidores a monitorear se instaló el plugin NRPE con el demonio extendido de internet Xinetd. El proyecto se basa en el monitoreo confiable de los servicios en los clientes para poder resolver problemas de manera rápida y de esta manera mantener los sitios funcionando de manera correcta.

1.7 Planteamiento del problema

Teniendo este panorama y contemplando que los servidores no se encuentran de manera física en las instalaciones de la empresa se puede apreciar que es de vital importancia el tener un monitoreo eficaz y confiable para atender problemas que los sistemas puedan presentar. El mantener un sitio no disponible por un periodo de tiempo prolongado representa una gran pérdida económica para la empresa. Todos los sitios de la empresa Grupo JOBO se encuentran alojados en Data Centers en USA llegando a ser más de 60 servidores rentados para el uso de los servicios requeridos para el funcionamiento de los sitios de manera correcta. Los principales Data Center en los que se rentan estos servidores son RackSpace y Digital Ocean. Por lo cual el no tener un monitoreo adecuado representa un gran problema para la empresa y sus proyectos.

1.8 Justificación

En el mundo actual, donde la tecnología es una herramienta muy útil para la vida cotidiana. La mayoría de todas las actividades que realizamos día con día de alguna u otra manera está ligada con los servicios en internet. Estas aplicaciones funcionan gracias a equipos especiales que ejecutan procesos de acuerdo al tipo de aplicación que alojan llamados servidores.

Sí un servidor llegase a fallar además de dejar de proveer sus servicios desencadena una serie de problemas mucho mayores si más servidores dependen de éste.
A tal grado de dejar sin funcionar a sitios, aplicaciones o sistemas completos. Es por ellos qué un monitoreo es de vital importancia en empresas que sus proyectos dependes de estos equipos.

Este tema se eligió de manera particular por el autor ya que la tendencia del mundo de la tecnología y de los sistemas informáticos tiende a una completa virtualización de aspectos como aplicaciones, software, y sistemas: por lo que es de suma importancia para un ingeniero en informática saber administrar y corregir problemas en sistemas basados en internet.

El monitoreo de los servicios que brindan los servidores para el correcto funcionamiento de los diferentes proyectos de Grupo JOBO ayuda principalmente mejorar la experiencia del usuarios final, haciendo que el usuario siempre tenga los sitios disponibles, que sus consultas sean rápidas, que los sitios carguen de manera rápida. ¿Con que herramientas se cuentan para el desarrollo?
Para el desarrollo de este proyecto se cuenta con un servidor dedicado exclusivamente para el alojamiento de la herramienta de monitoreo Nagios, herramienta la cual es gratuita. Una laptop para conexiones ssh con los servidores, y el plugin NRPE de igual manera sin ningún costo para instalar en los equipos clientes.

1.9 Objetivo general

Implementar un monitoreo de los servidores de la empresa Grupo JOBO con la herramienta Nagios y la funcionalidad de los sitios de la misma con Selenium.

1.9.1 Objetivos particulares

1. Instalar Nagios en el servidor dedicado para concentrar el monitoreo en el mismo.
2. Instalar el plugin NRPE en los equipos clientes junto con el demonio Xinetd para poder monitorear los servicios importantes de los hosts.
3. Crear monitoreos personalizados con scripts creados en bash.
4. Instalación de Selenium con el lenguaje Python para monitorear la funcionalidad de páginas web.
5. Programar las rutinas de monitoreo en Python para el monitoreo.
6. Ligar los resultados de Selenium con Nagios.

1.10 Alcances y limitaciones

1.10.1 Alcances:

Obtener un monitoreo confiable de los servidores y sus servicios pertenecientes a la empresa.

Dar atención y corrección a problemas de manera rápida recurrentes en los servidores.

Obtener un monitoreo completo además de automatizar tareas dependiendo del estado de los servicios.

Llevar a cabo un respaldo y monitoreo del mismo de los datos importantes de proyectos de la empresa.

1.10.2 Limitaciones:

Limitaciones en el rendimiento del hardware de los equipos, ya que el monitorear muchos servicios puede representar una carga y afectar el performance de los mismos.

Limitación en el hardware de la máquina donde se instale Selenium, ya que el ejecutar varias ventanas de un navegador puede ralentizar la máquina.

Capitulo II

METODOLOGÍA Y HERRAMIENTAS PARA EL DESARROLLO

2.1 Conceptos básicos

Estos conceptos estan sumamente relacionados con el proyecto desarrollado y descrito por este documento.

2.1.1 Internet

El internet (o, también, la internet) es un conjunto descentralizado de redes de comunicación interconectadas que utilizan la familia de protocolos TCP/IP, lo cual garantiza que las redes físicas heterogéneas que la componen funcionen como una red lógica única de alcance mundial. Sus orígenes se remontan a 1969, cuando se estableció la primera conexión de computadoras, conocida como Arpanet, entre tres universidades en California (Estados Unidos).

2.1.2 Arquitecturas lógicas de redes

La topología de una red es el arreglo físico o lógico en el cual los dispositivos o nodos de una red se interconectan entre sí sobre un medio de comunicación.

2.1.3 Cliente servidor

El modelo cliente servidor generalmente adopta la forma de un mensaje de solicitud del cliente hacia el servidor pidiendo que se efectúe alguna tarea específica. De esta forma el servidor recibe la petición, realiza la tarea y devuelve el resultado de ésta. Regularmente, muchos clientes utilizan un número pequeño de servidores, aunque esto depende del tipo de servicios que requiera el cliente para las tareas que realiza.

Un servidor es un nodo que forma parte de una red, provee servicios a otros nodos denominados clientes, estos clientes dependen de manera directa de los servicios y datos que proveen los servidores.

En este modelo un servidor atiende peticiones de los clientes, y además el servidor mismo puede ser cliente de algún otro servicio a la vez. El cliente envía en la petición los datos que requiere que sean procesados o analizados, el servidor los recibe y efectúa los procedimientos que estén definidos en él, los cuales dependen naturalmente de la naturaleza del servicio que presta en la red. Posterior a esto, regresa el resultado del procesamiento, y el cliente los recibe e interpreta a través de algún software c7omo navegadores web y así termina satisfactoriamente el modelo.

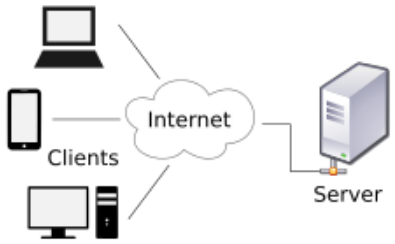

Figura 1.1 Modelo cliente – servidor

2.1.4 Maestro - esclavo

En este modelo de comunicación un dispositivo tiene un control unidireccional sobre uno o más dispositivo. Una vez que se ha establecido la relación de comunicación, el control siempre fluye desde el maestro hacia los esclavos. En ocasiones cuando varios esclavos no tienen un maestro, uno de ellos realiza la función del maestro temporalmente y cede el control al maestro cuando este está disponible.

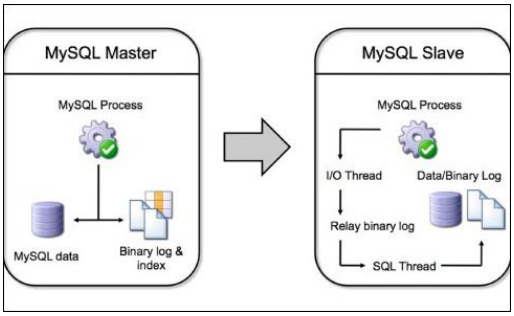

Figura 1.2 Modelo maestro – esclavo

2.1.5 Servicios en la nube

Los servicios de cómputo en la nube proporcionan tecnología de la información (TI) como un servicio a través de Internet o una red dedicada, con entrega según demanda

Figura 1.3 Servicios en la nube

y pago según el uso. Los servicios de cómputo en la nube abarcan desde plataformas de desarrollo y aplicaciones completas hasta servidores, almacenamiento y equipos de escritorio virtuales.

2.1.6 Monitoreo

Teniendo en cuenta los modelos bajos los cuales pueden trabajar los dispositivos en internet, es de gran importancia tomar en cuenta las características de hardware en los dispositivos ya que de ello depende el rendimiento de los mismos; rendimiento que se traduce en respuestas a peticiones, procesamiento de datos, soporte a múltiples conexiones y datos a almacenar. Otro aspecto a destacar es la seguridad de los dispositivos ya que nunca se ha de descartar los ataques malintencionados para así poder extraer información o simplemente causar algún daño en el rendimiento de los dispositivos.

El monitoreo de los recursos y de la seguridad de los dispositivos ayuda al personal de TI a mitigar problemas de causa natural o problemas causados con intención por personas ajenas o propias de la empresa.

2.2 Herramientas y especificaciones técnicas

Es importante destacar las características de los equipos donde residen las herramientas, debido a que estas se alojan junto con los proyectos y el monitoreo no debe intervenir con procesos de producción. Entonces por ende el hardware y configuración de los servers debe ser la más óptima posible.

2.2.1 Nagios

Nagios es un sistema de vigilancia o monitoreo de gran alcance que permite a las organizaciones identificar y resolver problemas de infraestructura de TI antes de que afecten los procesos de negocios críticos. Nagios es una herramienta poderosa que permite detectar y mitigar problemas antes de que afecten a los usuarios finales y clientes.

Entre sus características principales figuran la monitorización de servicios de red, la monitorización de los recursos de sistemas hardware, independencia de sistemas operativos, posibilidad de monitorización remota mediante túneles SSL cifrados o SSH. Nagios nos permite recibir alertas vía correo electrónico o mensajes SMS cuando alguno de los servicios monitorizados deja de funcionar o sobre pasa ciertos parámetros establecidos como puede ser el espacio libre en disco duro.

Mediante el uso de Nagios se puede:

- Crear un plan para la mejora de la infraestructura antes de que causen fallas los sistemas obsoletos.
- Responder a las emisiones en la primera señal de un problema.

- Reparar de manera inmediata los problemas cuando se detectan.
- Coordinar las respuestas del equipo técnico.
- Asegurar que cortes de infraestructura de TI tienen un efecto mínimo en los resultados de la organización.
- Control de toda la infraestructura y de los procesos de negocio.

Es por ello que se eligió esta poderosa herramienta y por la poca carga que representa en el procesador de los clientes.

2.2.2 Selenium

Es un entorno de pruebas de software para aplicaciones basadas en la web. Selenium provee una herramienta de grabar/reproducir para crear pruebas sin usar un lenguaje de scripting para pruebas (Selenium IDE). Incluye también un lenguaje específico de dominio para pruebas (Selanese) para escribir pruebas en un amplio número de lenguajes de programación populares incluyendo Java, C#, Ruby, Groovy, Perl, Php y Python. Las pruebas pueden ejecutarse entonces usando la mayoría de los navegadores web modernos en diferentes sistemas operativos como Windows, Linux y OSX. Se eligió Selenium debido a que se tenía un antecedente de este proyecto con behat mink pero debido a que es un sublenguaje de programación no terminaba las rutinas a tiempo y el NRPE marcaba un socket time out.

2.2.3 Lógica de respaldos

La lógica de respaldos en este proyecto tomó un aspecto muy importante debido a problemas con pérdida de información en los data centers. La lógica de respaldos hace referencia a la planeación coordinada de rutinas en bash para poder llevar a cabo un respaldo correcto además de planes de contingencia para todos los escenarios posibles, ya sean los óptimos o en caso contario los de gravedad.

Para crear un respaldo correcto hay varios aspectos y variantes a considerar, como lo son:

- Espacio de discos duros, tanto en el equipo donde está la información como el equipo a donde se va a respaldar.
- Qué procesos lleva el equipo a respaldar.
- Horarios para el respaldo donde no se interrumpan los procesos del servidor y esto afecte a los usuarios finales.
- Decidir quién llevará el proceso principal del respaldo.
- Frecuencia del respaldo.
- Cuántos respaldos se conservarán.
- De qué equipo se creará el respaldo.
- Cuánto costaría de manera mensual el respaldo (ya que los data centers como RackSpace cobran el ancho de banda de descarga).

- Procedimientos, esta parte es de vital importancia debido a que por ejemplo las bases de datos que funcionan con réplicas, no sólo se puede copiar el contenido de la carpeta contenedora de la base, sino parar el esclavo que para ello se toman en cuenta dos parámetros tanto del master como del esclavo, y posteriormente detener el servicio de MySQL. En respaldos de bases de datos es muy importante que se tomen en cuenta estos aspectos, de lo contrario se puede dañar la replicación y puede tomar hasta una semana de trabajo poder arreglar un problema de este tipo.

Se llevaron a cabo respaldos de páginas web completas, bases de datos completas, tablas de bases de datos, estructuras de bases de datos, estructuras de bases de datos, configuraciones, crons y repositorios.

2.2.4 Scripteo en bash.

Bash: Bourne again Shell, es un programa informático, cuya función consiste en interpretar órdenes, y un lenguaje de programación de consola. Está basado en la shell de Unix y es compatible con POSIX.

Por políticas de la empresa, todos los equipos de cómputo dentro y fuera de ella trabajan con distribuciones de Linux, como Ubuntu en equipos de la empresa, CentOS en los vps, y RedHat en equipos de red. Debido a lo anterior los aspectos de monitoreo como plugins y respaldos se crearon con rutinas en bash. Los comandos más importantes a resaltar de bash y los que más se utilizaron fueron:

- grep: buscador de texto en archivos, suena simple pero es un potente procesador de textos, ya que busca, remplaza, elimina, da formato a las salidas, concatena, busca patrones, usa expresiones regulares y demás.
- Awk: tiene la misma función que grep pero se diferencia en que awk puede trabajar con más de una línea al mismo tiempo además de que puede usar expresiones regulares mucho más complejas que grep.
- Ls: despliega la lista de archivos contenidos en la carpeta donde se aplica, suena simple pero los valores que agregan sus modificadores la hace una herramienta muy potente.
- Du: regresa el peso del archivo en cuestión.
- Scp: secury copy, hace una copia de archivos por ssh.
- Rsync: sincroniza archivos, es decir en respaldos ya creados solo sincroniza los archivos faltantes, todo ello por ssh.
- Cat, more, less, tail: lectores de archivos con variantes entre ellos.
- Tar: compresión y descompresión de archivos.

Con estos pocos pero poderosos comandos se realizó el scripteo para plugins y para las lógicas de respaldos.

2.2.5 Editor VIM

Para la edición de archivos en todos los aspectos de este proyecto se utilizó vim. Un potente editor de texto para Linux y que supone una gran ventaja a la hora de trabajar debido a que no se usa el mouse para poder editar, todo se lleva a cabo con comandos.

El no contar con una interfaz gráfica supone una desventaja cuando se comienza a utilizar.

2.2.6 Xinetd

La principal función de Xinetd es la conectividad y acceso a los servicios en los equipos instalados, siendo Xinetd la principal vía de comunicación entre el servidor donde se aloja Nagios y los hosts con NRPE. Dicha comunicación hace el intercambio de la información necesaria para poder monitorear el estado de los servicios. La información que Nagios necesita para poder interpretar las salidas, que pueden ser: un ok dado por un exit 0, un exit 1 para un warning, o bien un exit 2 para un critical siendo este el estado menos deseado en un servicios ya que representa un mal funcionamiento del servicio y un problema en el funcionamiento de los demás servicios dependientes de él.

2.3 Servicios ejecutados dentro de los servidores remotos

- MYSQL
- MySQL es un sistema de gestión de bases de datos relacional, multihilo y multiusuario con más de seis millones de instalaciones. MySQL AB —desde enero de 2008 una subsidiaria de Sun Microsystems y ésta a su vez de Oracle Corporation desde abril de 2009— desarrolla MySQL como software libre en un esquema de licenciamiento dual.
- MongoDB
- MongoDB (de la palabra en inglés “humongous” que significa enorme) es un sistema de base de datos NoSQL orientado a documentos, desarrollado bajo el concepto de código abierto. MongoDB forma parte de la nueva familia de sistemas de base de datos NoSQL.
- En vez de guardar los datos en tablas como se hace en las base de datos relacionales, MongoDB guarda estructuras de datos en documentos tipo JSON con un esquema dinámico (MongoDB llama ese formato BSON), haciendo que la integración de los datos en ciertas aplicaciones sea más fácil y rápida.
- ElasticSearch
- Elasticsearch es un servidor de búsqueda basado en Lucene. Provee un motor de búsqueda de texto completo, distribuido y con capacidad de multi-tenencia con una interfaz web RESTful y con documentos JSON. Elasticsearch está desarrollado en Java y está publicado como código abierto bajo las condiciones de la licencia Apache.
- Apache
- El servidor HTTP Apache es un servidor web HTTP de código abierto, para plataformas Unix (BSD, GNU/Linux, etc.), Microsoft Windows, Macintosh y otras, que implementa el protocolo HTTP/1.1 y la noción de sitio virtual.
- Redis
- Redis es un motor de base de datos en memoria, basado en el almacenamiento en tablas de hashes (clave/valor) pero que opcionalmente puede ser usada como una base de datos durable o persistente. Está escrito en ANSI C por Salvatore Sanfilippo quien fue patrocinado por VMware y, a partir de 2013, por Pivotal Software. Está liberado bajo licencia BSD por lo que es considerado software de código abierto.
- Memcached
- Memcached es un sistema distribuido de propósito general para caché basado en memoria, diseñado por Danga Interactive y que es muy usado en la actualidad por múltiples sitios web. Memcached es empleado para el almacenamiento en caché de datos u objetos en la memoria RAM, reduciendo así las necesidades de acceso a un origen de datos externo (como una base de datos o una API).

Capítulo 3

Monitoreo usando Nagios, NRPE y Selenium además de lógicas y monitoreos de respaldos.

En este capítulo se describe el implantamiento del proyecto, desde la instalación de las herramientas (Nagios, NRPE, Selenium), y la configuración en las mismas para poder llevar a cabo los monitoreos. Así como también las especificaciones de hardware de los equipos donde se alojan las herramientas ya mencionadas.

Primeramente se especifican las características de hardware de los servidores y para qué son usados. Para posteriormente proceder a los pasos de instalación de las herramientas. Finalmente se documentan las configuraciones que se siguieron para poder dar de alta los monitoreos tanto de servicios de los servidores como de la funcionalidad de las páginas web de la empresa y de los respaldos que se ejecutan de manera automatizada para verificar que se creen de manera correcta, o de lo contario corregir los mismo de manera inmediata.

También se describe la programación de las rutinas más importantes desarrolladas durante este proyecto de monitoreo.

3.1 Nagios

Se eligió Nagios ya que los servidores son la principal fuente de producción de la empresa, por lo cual estos deben de tener un rendimiento y condiciones óptimas para dar un servicio agradable al usuario final. Dado que Nagios es una herramienta muy poderosa, configurable y liviana en procesos de ejecución en los clientes remotos, para este proyecto se instaló Nagios Core 3.2.0.

3.1.1 Servidor dedicado Nagios

Se optó por rentar un servidor dedicado exclusivamente para el alojamiento de la herramienta Nagios debido a la cantidad de monitoreos a configurar, y por la carga que suponen estos procesos no sería posible alojar Nagios en un servidor de producción.

Las especificaciones de hardware del servidor dedicado son:

- Procesador Intel Xeon de 11 núcleos a 2.4 GHz.
- 1 GB en memoria RAM.
- 200 GB en disco duro.
- Sistema operativo CentOS en su versión 5.11 instalado en modo texto.
- Sin memoria Swap

Con estas características se asegura que el servidor dedicado de Nagios tendrá la capacidad y los recursos suficientes para completar todos los procesos de peticiones de los plugins que se lanzan cada periodo de actualización.

Actualmente el servidor destinado para Nagios lleva a cabo más de 700 monitoreos con intervalos de actualización de 30 segundos haciendo que esta carga de procesos se dispare en manera de ráfaga teniendo como repercusiones la pérdida para ciertos plugins que llevan a cabo varios procesos complejos.

3.1.2 Servidores remotos o clientes a monitorear

Los servidores a monitorear varían en cuanto a características debido a que los proyectos y servicios que alojan dependen de los fines de los mismos.

- Procesadores Intel Xeon de 2 a 8 núcleos.
- De 1 a 4 GB's en memoria RAM.
- Discos duros de capacidades de entre 50 a 500 GB's dependiendo el tipo de disco duro (sólidos o mecánicos).
- Memorias Swap de entre 1 a 10 GB's.

3.2 Instalación

Las instalaciones de las herramientas son paso importante ya que a partir de ellos se conoce el funcionamiento de las mismas y así poder resolver problemas de funcionamiento en un futuro.

3.2.1 Instalación de Nagios

Prerrequisitos: para la instalación de Nagios se requiere la versión más reciente de apache, y los siguientes paquetes.

- Gcc
- Glibc
- glibc-common
- gd
- gd-devel
- ssl-headers

De no contar con éstos se deben instalar.

1.- Una vez asegurado que contemos con los prerrequisitos procedemos a crear un usuario para el manejo de Nagios. Esta acción se ejecuta con privilegios de administrador de lo contrario generará un error.

- useradd –m Nagios

2.- Generamos el grupo nagcmd para permitir el envío de comandos desde la consola web y agregamos los usuarios nagios y

- apachegroupadd nagcmd
- usermod -a -G nagcmd nagios
- usermod -a -G nagcmd apache

3.- Generamos la carpeta donde descargaremos Nagios

- mkdir /opt/Nagios

4.- Descargamos dentro de nuestra nueva carpeta: Nagios core 3.2.0

- wget http://prdownloads.sourceforge.net/sourceforge/nagios/nagios-3.2.0.tar.gz

Nota, el comando wget no está instalado por default en los sistemas Linux, sino lo tiene instalado realizar la instalación con el comando *yum install wget*.

5.- También descargamos los plugins oficiales: Nagios Plugins 1.4.14

- Wge thttp://prdownloads.sourceforge.net/sourceforge/nagiosplug/nagios-plugins-.4.14.tar.gz.

6.- Descomprimimos la carpeta de Nagios

- cd /opt/Nagios
- tar xzf nagios-3.2.0.tar.gz
- cd nagios-3.2.0

7.- Configuramos y Compilamos los archivos de instalación

- ./configure --with-command-group=nagcmd
- make all
- Compilamos binarios, init script y archivos de ejemplos de configuración
- make install
- make install-init
- make install-config
- make install-commandmode

Nota: con esto nagios queda instalado en /usr/local/Nagios

8.- Instalación de interfaz Web y Creación de Usuario Admin

- cd /opt/nagios/nagios-3.2.0
- make install-webconf
- htpasswd -c /usr/local/nagios/etc/htpasswd.users nagiosadmin
- service httpd restart

La contraseña ingresada en este paso se requiere más tarde para ingresar a la interfaz web

9.- Instalación y compilación de Nagios Plugins, los plugins son comandos precargados, dentro de los mismo existen comandos para el monitoreo del espacio en disco duro, carga de procesos, memoria Ram libre, verificación del servicio apache.

Descomprimimos:

- cd /opt/Nagios
- tar xzf nagios-plugins-1.4.14.tar.gz
- cd nagios-plugins-1.4.14

Configuramos y Compilamos

- ./configure --with-nagios-user=nagios --with-nagios-group=Nagios
- make
- make install

10.- Verificamos los archivos de configuración

- /usr/local/nagios/bin/20agios –v /usr/local/nagios/etc/nagios.cfg

Y debemos tener una salida en las últimas líneas como la siguiente:

- Total Warnings: 0
- Total Errors: 0

11.- Permitimos que Nagios y apache inicien al iniciar nuestro sistema

- chkconfig --add Nagios
- chkconfig nagios on
- chkconfig httpd on

Iniciamos Nagios

- service nagios start

A partir de este punto Nagios está instalado en el servidor, y se puede acceder a la interfaz web de Nagios escribiendo en el navegador http://Ip-Server/nagios/

(Imagen de login Nagios)

(Imagen de servicios)

2.4.2 Instalación de NRPE

Se instaló el plugin NRPE en su versión 1.4.14

1.- Instalamos Openssl, librearía necesaria para poder compilar los archivos a instalar.

- yum install openssl-devel

2.- Generamos el usuario Nagios, con permisos de administrado (root)

- /usr/sbin/useradd Nagios
- passwd Nagios

3.- Descargamos y Descomprimimos Nagios Plugins

- mkdir /opt/Nagios
- cd /opt/Nagios
- wget http://prdownloads.sourceforge.net/sourceforge/nagiosplug/nagios-plugins-1.4.14.tar.gztar
- nagios-plugins-1.4.14.tar.gz

4.- Compilamos e Instalamos Nagios Plugins

- cd nagios-plugins-1.4.14
- ./configure
- make
- make install

5.- Cambiamos Permisos

- chown nagios.nagios /usr/local/Nagios
- chown -R nagios.nagios /usr/local/nagios/libexec

6.- Instalamos Xinetd

- yum install Xinetd

7.- Descargamos y Descomprimimos el Demonio NRPE

- cd /opt/Nagios
- wget http://prdownloads.sourceforge.net/sourceforge/nagios/nrpe-2.12.tar.gz
- tar xzf nrpe-2.12.tar.gz
- cd nrpe-2.12

Compilamos e Instalamos NRPE

- ./configure
- make all
- make install-plugin
- make install-daemon
- make install-daemon-config

8.- Instalamos el Demonio NRPE Como un Servicio en Xinetd

- make install-xinetd

9.- Editamos el archivo /etc/xinetd.d/nrpe para agregar la ip de nuestro servidor Nagios

- vi /etc/xinetd.d/nrpe
- only_from = 127.0.0.1 <direccion-nagios>

10.- Agregamos la siguiente línea para nuestro demonio NRPE en el archivo /etc/services

- vi /etc/services
- nrpe 5666/tcp # NRPE

11.- Reiniciamos Xinetd

- service xinetd restart

12.- Probamos el demonio NRPE localmente

- netstat -at | grep nrpe

Debemos obtener la siguiente salida:

- tcp 0 0 *:nrpe *:* LISTEN

13.- Verificamos que el demonio NRPE está correctamente instalado, para ello utilizaremos el plugin check_nrpe que instalamos para pruebas

- /usr/local/nagios/libexec/check_nrpe -H localhost

Debemos de obtener la siguiente salida:

- NRPE v2.12

3.3 Configuración

La configuración de los parámetros en Nagios y dentro del NRPE es un paso muy importante, ya que a partir de este configuración se empieza el monitoreo de host remotos, y depende de nuestra configuración que todo funcione de manera correcta.

Dentro de este apartado se tocan temas muy importantes ya que, se aborda la estructura y el funcionamiento de Nagios con el NRPE. Se describen las carpetas y archivos de configuración principales para poder llevar a cabo el monitoreo de host remotos.

3.3.1 Carpetas y archivos de configuración importantes de Nagios y del NRPE.

Nagios: del lado de Nagios la carpeta de configuración principal se encuentra en */usr/local/nagios/.*

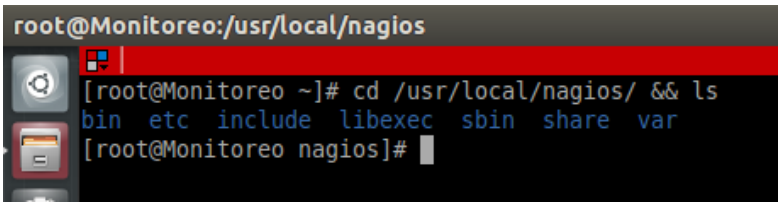

Figura 2.1 Directorio /usr/local/nagios

De las carpetas que figuran en la imagen anterior, las más relevantes son:

Etc: contiene:

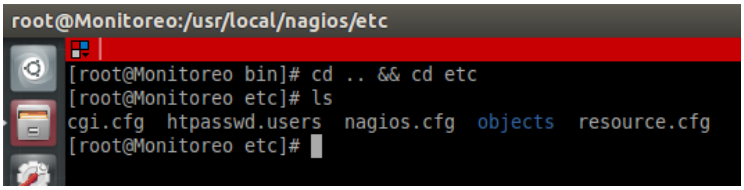

Figura 2.2 Directorio /usr/local/nagios/etc

El archivo cgi.cfg contiene las rutas relativas o paths hacia los archivos de configuración usados para el monitoreo, aquí se definen los nuevos archivos de host remotos, también en este archivo se define el intervalo de tiempo en el cual Nagios actualiza los estados de sus monitoreos:

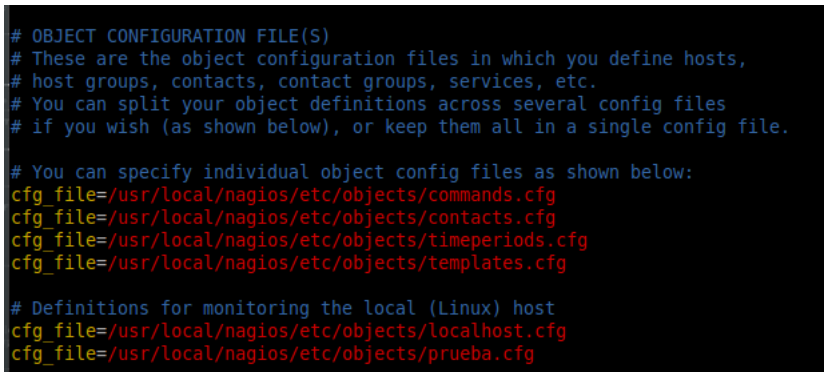

Figura 2.3 Archivo cgi.bin

El archivo htpasswd.users contiene la contraseña que se le definió en la instalación al usuario nagios. El archivo nagios.cfg es un archivo de compilación para cuando se modifican monitoreos o se levantan nuevos. El archivo resources.cfg es un archivo cifrado propio de Nagios.

La carpeta objects contiene los archivos principales de configuración:

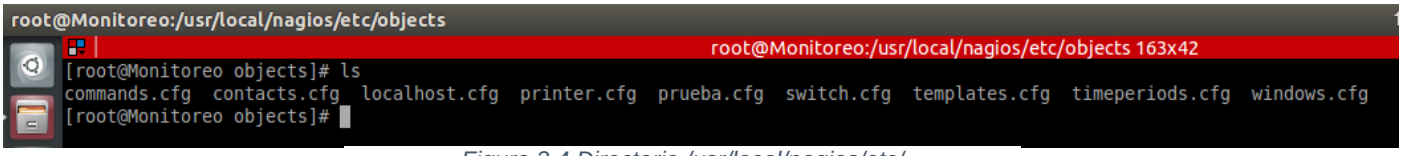

Figura 2.4 Directorio /usr/local/nagios/etc/

Estos archivos son muy importantes ya que de su correcta configuración depende del funcionamiento de Nagios. Y sus funciones son:

Commands.cfg: este archivo define comandos con una estructura definida, estos comandos se aplican a los host remotos, los cuales tiene que tener definido el mismo comando y estar ligado a un plugin.

```
define command{
        command_name    check_hda1
        command_line    $USER1$/check_nrpe -H $HOSTADDRESS$ -c check_hda1
        }
```

Figura 2.5 Archivo commands.cfg

Contacts.cfg: este archivo define los contactos a los cuales se les va a alertar en situaciones de riesgo en los servicios, se puede alertar por sms configurando un Gateway de sms's o por mail. Se definen contactos de manera individual y además de agregan a grupos para una mejor y fácil administración, y esto se explica más adelante.

```
# Just one contact defined by default - the Nagios admin (that's you)
# This contact definition inherits a lot of default values from the 'g
# template which is defined elsewhere.

define contact{
        contact_name                    nagiosadmin             ; Shor
        use                             generic-contact         ; Inhe
        alias                           Nagios Admin            ; Full

        email                           auxsistemasjobomas@gmail.com
        }
```

Figura 2.6 Archivo contacs.cfg

Los archivos localhost, prueba, printer, switch, y Windows todos con terminación .cfg son archivos para monitorear host remotos y locales como el caso del monitoreo propio en el archivo localhost.

Templates.cfg: define las platillas para cada cuestión de los parámetros de monitoreo, como lo son los contactos, hosts, grupos, tipos de hosts etc.

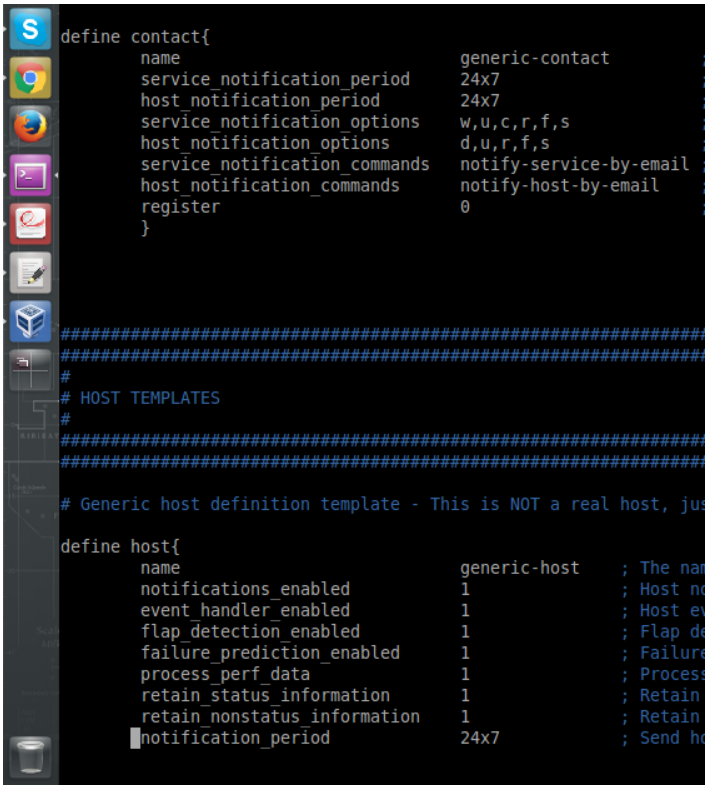

Figura 2.7 Archivo templates.cfg

Timeperiods.cfg: este archivo define los tiempos en los cuales el monitoreo va a estar funcionando de manera completa, ejemplo se define un horario de 24x7, esto quiere decir que el monitoreo es de manera intensa con sms's e e-mails todas las horas todos los días de la semana. Por el contrario si se definen horarios de trabajo el monitoreo

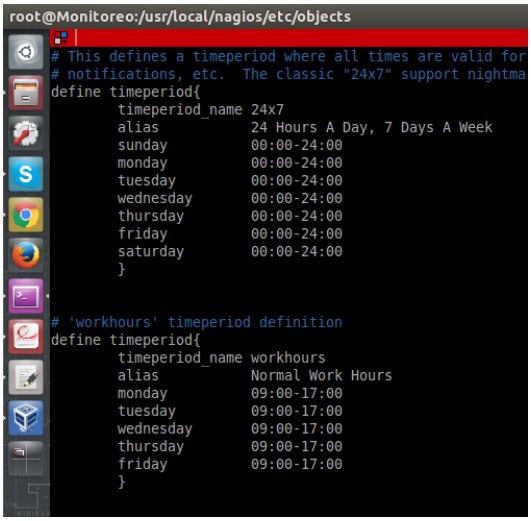

Figura 2.8 Archivo timeperiods.cfg

sólo está de manera completa en horarios laborales y después funciona sólo el monitoreo web.

Libexec dentro de */usr/local/nagios* contiene los plugis o scripts que ejecutan los comandos definidos en commands.cfg o en el nrpe.cfg.

```
[root@Monitoreo libexec]# ls
check_apt       check_disk      check_ftp       check_ircd      check_nagios    check_nwstat    check_ram       check_swap      check_wave
check_breeze    check_disk_smb  check_http      check_load      check_nntp      check_oracle    check_real      check_tcp       negate
check_by_ssh    check_dns       check_icmp      check_log       check_nrpe      check_overcr    check_rpc       check_time      selenium.back
check_clamd     check_dummy     check_ide_smart check_mailq     check_nt        check_ping      check_selenium_jobo check_udp   urlize
check_cluster   check_file_age  check_ifoperstatus check_mrtg   check_ntp       check_pop       check_sensors   check_ups       utils.pm
check_dhcp      check_flexlm    check_ifstatus  check_mrtgtraf  check_ntp_peer  check_procesos  check_smtp      check_uptime    utils.sh
check_dig       check_fping     check_imap      check_mysql     check_ntp_time  check_procs     check_ssh       check_users
[root@Monitoreo libexec]#
```

Figura 2.9 Directorio /usr/local/nagios/lib

Dentro de la carpeta /usr/local/nagios existe la carpeta share, carpeta donde se aloja los archivos de la página web de monitoreo, que tiene un aspecto al siguiente:

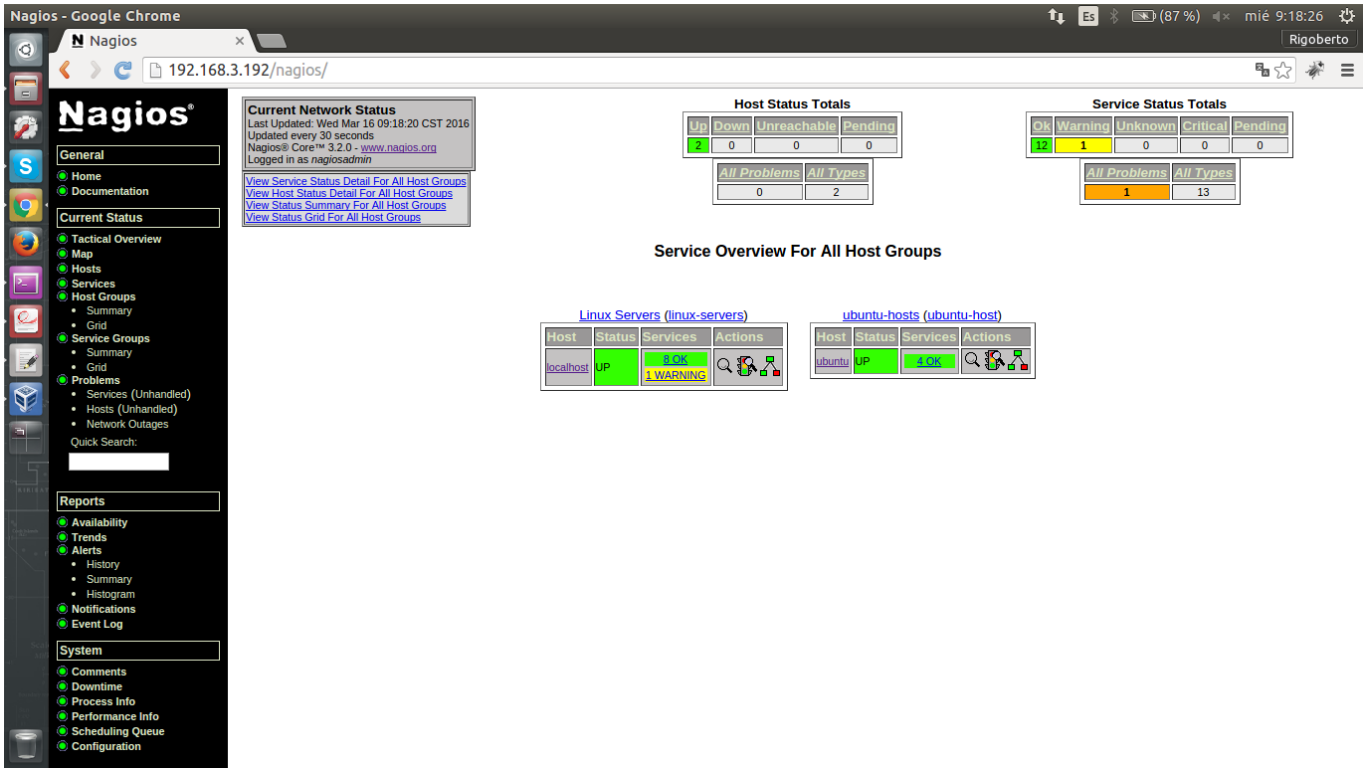

Figura 2.10 2.10 Web Nagios

Las carpetas sbin y var contienen archivos cifrados propios de Nagios.

NRPE: del lado del NRPE de igual manera existe el path */usr/local/nagios/* sólo que no contiene todos los archivos que contiene el servidor dedicado:

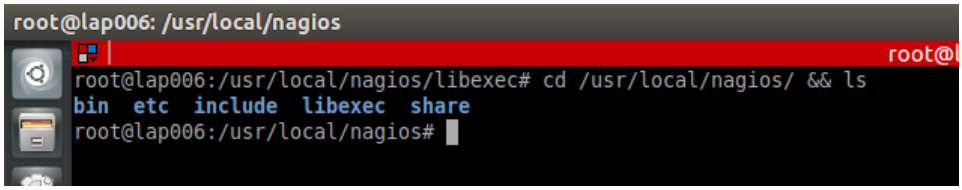

Figura 2.11 Directorio /usr/local/nagios

Dentro de este directorio las carpetas que tiene contenido importante son etc, que contiene el archivo nrpe.cfg: el cual define los comandos y el script o plugin que ejecuta el comando.

Libexec que contiene los plugin o scripts que ejecutan los comandos:

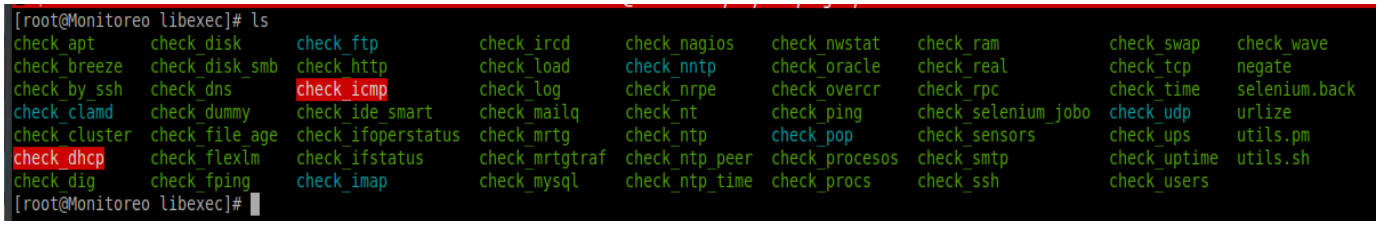

Figura 2.12 Directorio /usr/local/nagios/libexec

3.4 Monitoreo de un host remoto

En esta parte se pone en funcionamiento el monitoreo de los host remotos. Los pasos para dar de alta un host en Nagios son:

3.4.1 Del lado de NRPE

1. Scripteo del plugin o uso de los previamente instalados.

Para este caso se usa un script realizado en bash con la siguiente estructura, que debe estar creado o localizado en /usr/local/nagios/libexec/ y que para este ejemplo llamaremos check_ram:

```
#!/bin/bash
info=$(free -m | grep buffers/cache)
ramlibre=$(echo $info | awk '{print $4}')
ramusada=$(echo $info | awk '{print $3}')
echo "Memoria Ram Usada:$ramusada Mb,Libre: $ramlibre Mb"
if [ $ramlibre -le 259 ],
then
```

```
        exit 2
elif [ $ramlibre -le 900 ] && [ $ramlibre -ge 260 ]
then
        exit 1
else
        exit 0
fi
```

El script anterior lo que hace es recuperar la información de la memoria ram con: info=$(free -m | grep buffers/cache) , que lo que hace es ejecutar el comando free que como salida nos da la información de la memoria ram libre y ocupada del sistema en megas debido al modificador -m, con algunas otras líneas que no nos interesan por lo que por una tubería representada por el símbolo | (pipe en inglés) se pasa la salida del comando completa y con el comando grep recorta sólo la línea que coincida con la línea “buffers/cache”, que es la línea que contiene la información de la ram del sistema y se guarda en la variable info. Posterior a esto la línea ramlibre=$(echo $info | awk '{print $4}') trabaja sobre la variable $info y su contenido imprimiéndola para posteriormente recortar la columna $4 obteniendo así la memoria ram libre. La línea ramusada=$(echo $info | awk '{print $3}') de igual manera hace un echo sobre la variable $info y recorta la columna 3 para así obtener el total en megas de memoria ram usada.

La linea echo "Memoria Ram Usada:$ramusada Mb,Libre: $ramlibre Mb" imprime la información de los anteriores comandos y cadenas descriptivas, dichas cadenas con las que se imprimirán en el alertamiento de la web de Nagios. Esta línea se puso antes de los if's por optimización de código ya que ahorra líneas de código y procesamiento además de que dicha información se imprime cual sea el resultado que se obtenga.

El primer if hace una comparación sobre $ramlibre sí es menor o igual a 259 manejando las cantidades en megas, sí esta comparación es verdadera hace un exit 2 que es la señal por la cual Nagios sabe que el servicio o en este caso la ram tiene poco espacio para ser usado y que podría provocar fallos en el sistema. O bien sino pasamos a un elseif donde comparamos si la memoria ram libre cae en el rango de mayor o igual a 256 y menor o igual a 900 donde si se cumple hacemos un exit 1 con lo cual indicamos a Nagios que es un warning y que las cosas podrían o no pasar a ser críticas. De lo contrario hacemos un exit 0 que indica a Nagios que todo está bien y que tenemos memoria ram suficiente para seguir trabajando.

Los monitoreos de este tipo ya sean ram, disco duro, carga de procesador, nivel de procesos etc, depende de las características en hardware del servidor en cuestión, ya que no se aplican las mismas reglas de comparación para un servidor con 2GB en RAM que a uno de 8.

2. Comprobación del funcionamiento del script o plugin de forma local.

La comprobación del script de forma manual o local se hace ejecutando el script, primeramente se le dan permisos de ejecución al archivo con el comando chmod +x “nombre_archivo”, y para corroborar que efectivamente haga lo que nosotros necesitamos ejecutamos el script con ./”nombre_archivo” o bien sh “nombre_archivo”, una vez que nuestro script es funcional podemos pasar al siguiente paso.

3. Definir el comando en el archivo /usr/local/nagios/etc/nrpe.cg

Para agregar el comando al NRPE editamos el archivo /usr/local/nagios/etc/nrpe.cfg, buscamos el bloque donde están los comandos y agregamos la linea con la siguiente estructura:

- command[“nombre del comando”]=/ruta/al/script/

Para nuestro ejemplo la estructura quedaría de la siguiente manera:

- command[check_ram]=/usr/local/nagios/libexec/check_ram

Y con esto el comando check_ram ya está dado de alta dentro de NRPE

4. Comprobar que el NRPE comprenda el plugin

Comprobar que el NRPE comprenda el plugin es una parte muy importante debido a que si por alguna razón el intérprete del NRPE no puede procesar las salidas de nuestro script nos causará un error. Por lo cual para comprobar que el NRPE comprenda en script ejecutamos:

- /usr/local/nagios/libexec/check_nrpe -H 127.0.0.1 -c check_ram

Con lo cual le estamos diciendo que use el plugin check_nrpe que es el intérprete de Nagios y del NRPE para ejecutar sobre un host (modificador -H), la dirección del host que para este caso es el localhost, el comando (modificador -c) check_ram previamente dado de alta dentro del archivo nrpe.cfg. Sí no hay ningún problema en cuanto a nuestra configuración o con el intérprete deberíamos obtener la misma salida o resultado que si ejecutáramos el script de forma manual.

Problemas recurrentes: los problemas recurrentes cuando no obtenemos las salidas esperadas son;

- Que el nombre del archivo difiera con el que se da de alta en el nrpe.cfg
- Que el comando tenga una ruta equivoca hacía el archivo a ejecutar.
- Que el archivo no tenga permisos de ejecución.
- Que el directorio donde se encuentra el script no esté ligado al grupo del usuario Nagios (por ello es recomendable almacenar los scripts en libexec).
- Que el NRPE no comprenda la salida del script, ya sea por palabras reservadas o por problemas de lectura de archivo al no poner las rutas relativas completas.

De este modo cuando exista un problema en algún alertamiento procederemos a diagnosticar el fallo de manera más rápida debido a que nos aseguraríamos que el plugin funciona de manera correcta.

3.4.2 Del lado de Nagios

1. Dar de alta el host

Para dar de alta el host a monitorear lo correcto es crear un archivo con terminación .cfg y un nombre alusivo a la función del servidor, o agregar el host a un archivo ya existente; por default Nagios contiene el archivo localhost.cfg que sirve a Nagios para monitorearse a sí mismo pero lo recomendable es crear un archivo separado.

Dentro del archivo donde sea que se agregue el host se tendrá que definir de la siguiente manera:

```
define host{
        use                     linux-server
        host_name               localhost
        alias                   localhost
        address                 127.0.0.1
        }
```

Los parámetros que figuran en la sintaxis anterior significan:

- use: define la platilla referida a los hosts a utilizar, en dicha plantilla se define.
- host_name: se refiere al nombre específico con el que se hará referencia a dicho host.
- alias: define un alias que solo sirve como descripción del host.
- address: dirección ip publica del host remoto.

2. Agregar el host a un grupo

Agregar el host a un grupo es parte de una administración eficaz y fácil, ya que cuando se trata de monitorear un servicio de host que perteneces a un mismo grupo en vez de hacer referencia de host por host de manera individual se referencia al grupo y tiene la misma funcionalidad.

```
define hostgroup{
        hostgroup_name      linux-servers
        alias               Linux Servers
        members             localhost
        }
```

Para esta sintaxis los parámetros significan:

- hostgroup_name: el nombre con el que se referencia al grupo de host.
- alias: nombre informativo o descriptivo del grupo, ejemplo MySQL; esto hace referencia a que los servidores pertenecientes a ese grupo ejecutan MySQL.
- Members: es el campo donde se definen que hosts serán miembros del grupo, cada host se refeencia por su host_name y si son más de uno se separan por una coma entre ellos, ejemplo: members: localhost,hostremoto1,hostremoto2 … etc.

3. Agregar los comandos para monitorear para monitorear los host

Se deben definir los comandos dentro del archivo commands.cfg para poder enviar las peticiones del comando y tanto del lado de Nagios como del NRPE se deberán llamar de la misma manera, ejemplo:

```
define command{
   command_name check_ram
  command_line $USER1$/check_nrpe -H $HOSTADDRESS$ -c check_ram
}
```

La sintaxis es simpre:

- command_name: es el nombre del comando a ejecutarse.
- command_line: es la sintaxis del comando, tomando como parámetros variables de macros de Nagios, $USER1 igual al usuario de Nagios, se dispara la petición con el plugin check_nrpe que sirve como intérprete de la respuesta, más el modificador -H para indicar el host pasando como parámetro $HOSTADDRESS$ que recupera la dirección ip del host donde se aplica el comando, modificador -c que indica el comando a ejecutar de manera remota y se pasa como parámetro el nombre del comando.

4. Definir servicios para el host.

Se define qué servicios se van a monitorear sobre el host sobre el cual se está trabajando en este ejemplo el localhost, este es el último paso ya que en este paso se hace uso de todas las configuraciones anteriores.

```
define service{
        use                             local-service
        host_name                       localhost
        service_description             Ram
       check_command                    check_ram
}
```

Los parámetros de la sintaxis anterior son:

- use: que indica que plantilla para el servicio usar
- host_name: indica el host sobre el cual se va a ejecutar el comando.
- service_description: descripción de lo que el comando monitorea.
- check_command: indica el nombre del comando que se va a mandar a ejecutar de manera remota, dicho comando debe estar definido tanto en el archivo commands.cfg como del lado del NRPE en el archivo nrpe.cfg dentro de /usr/local/nagios/etc.

5. Comprobación de errores

Ya por último se hace una comprobación de errores para asegurarnos que estamos agregando el monitoreo del nuevo host de manera correcta y que no traerá problemas en la configuración de los hosts ya existentes, se emite el comando:

- /usr/local/nagios/bin/nagios -v /usr/local/nagios/etc/nagios.cfg

Lo cual nos hace una comprobación de errores y nos informa donde están en caso de existir, o de lo contrario nos dice que no se detectan errores en la comprobación. Ya para finalizar se reinicia Nagios para poder agregar nuestro nuevo host a la web y agregarlo a nuestra configuración global de Nagios, para ello se emite el comando:

- service nagios restart, o bien /etc/init.d/nagios restart.

Y en el entorno web de manera inmediata obtenemos un ítem en estado pending de color gris esperando al siguiente chequeo para definir su estado.

3.4.3 Interfaz web de Nagios

Login:

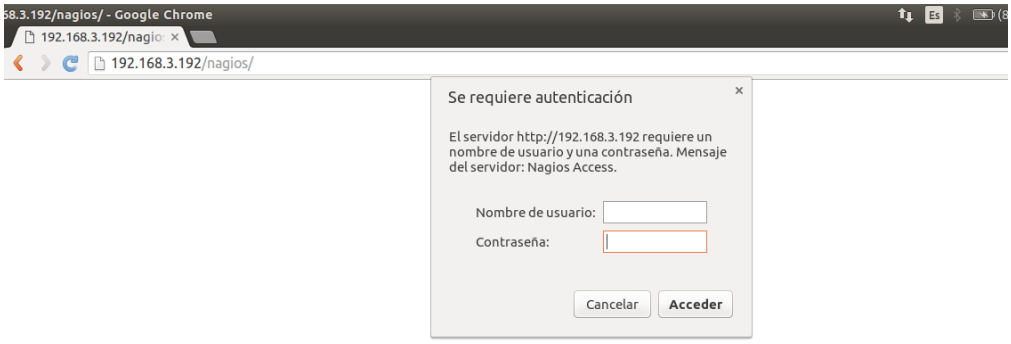

Figura 2.13 Login Nagios

Página de inicio:

Menú:

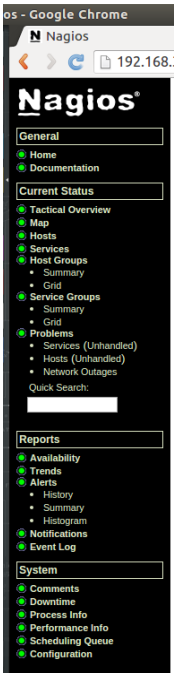

Figura 2.14 Menú Nagios

Información de hosts:

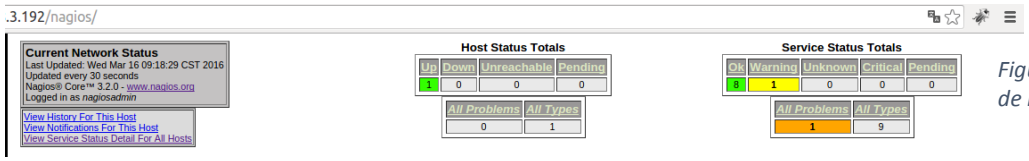

Figura 2.15 Información de host

Detalles de hosts:

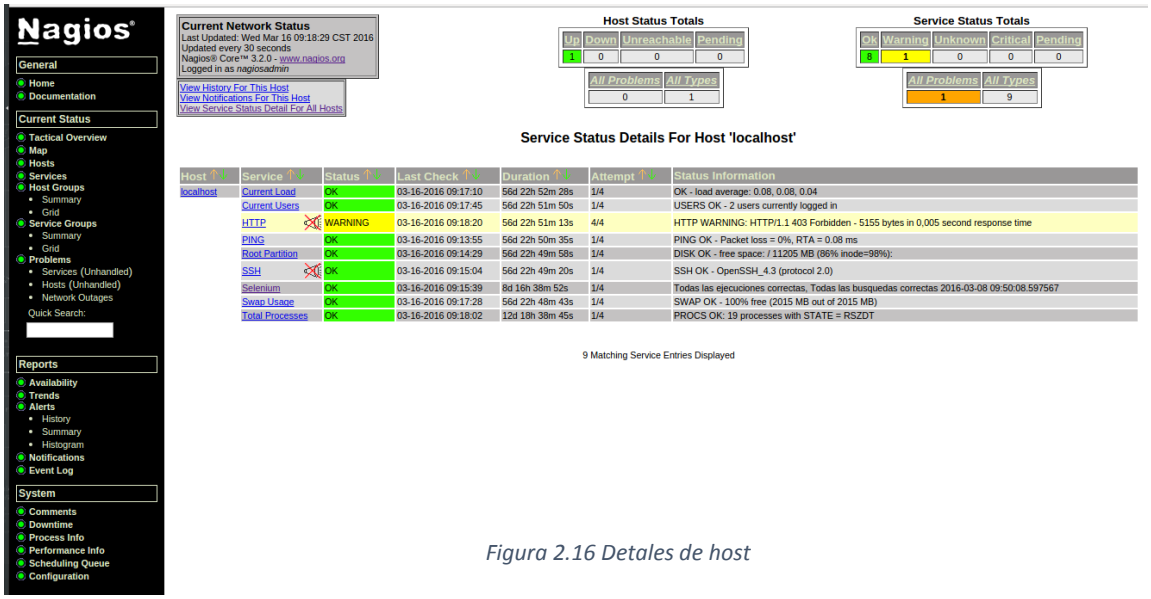

Figura 2.16 Detales de host

3.5 Monitoreo de páginas web con Selenium

Como ya se mencionó Selenium es un entorno de pruebas para aplicaciones web, esto quiere decir que se pueden programar rutinas automatizadas para poder simular un usuario real, para simular el clickeo de links, botones, apertura de páginas, reproducción de video, navegación entre menús etc. Todo ello con el fin de encontrar bugs, hacer mejoras y testeos ya sea de funcionalidad, estrés etc.

Para este proyecto Selenium se utilizó para hacer testeos de la disponibilidad de las páginas web de la empresa y arrojar los resultados a Nagios para así atender errores.

3.5.1 Instalación de Selenium con Python

Se instaló Selenium en una máquina con Ubuntu 15.10, con 4 Gb en RAM, 750 en disco duro mecánico. Se usó Firefox como navegador para las pruebas de Selenium.

Para instalar Selenium con Python como leguaje de programación se ejecuta el comando:

- pip install selenium

Y con ello Selenium estará instalado y listo para usarse. Primero se hace una prueba del driver con el siguiente código:

```
from selenium import webdriver
from selenium.webdriver.common.keys import Keys

driver = webdriver.Firefox()
driver.get("http://www.python.org")
assert "Python" in driver.title
elem = driver.find_element_by_name("q")
elem.send_keys("pycon")
elem.send_keys(Keys.RETURN)
assert "No results found." not in driver.page_source
driver.close()
```

Para este básico ejemplo pero que muestra lo potente que es esta herramienta hace lo siguiente. Importa el webdriver de Selenium así como también las librerías de las teclas para simular el envío de texto mediante el teclado.

Posterior a esto se instancia el navegador Firefox con la línea: `driver = webdriver.Firefox()`, posterior a esto con el objeto driver que contiene la instanciación del navegador, se le pasa la URL a recuperar. Una vez abierta la página se intenta hacer un match o encontrar una cadena en el texto del título (`assert "Python" in driver.title`). Se encuentra un elemento del código de la página con el nombre “q”, sí se encuentra el elemento se le pasa la cadena “pycon” de lo contrario se lanza un error. Y por último de cierra el navegador.

3.5.2 Lógica del monitoreo con Selenium

En total se monitorearon 5 servidores web esclavos, para comprobar las replicaciones del master, y así corregir errores.

La lógica que se manejó para poder arrojar los resultados más detallados posibles dentro de la web de Nagios es la siguiente y se explica en los siguientes temas:

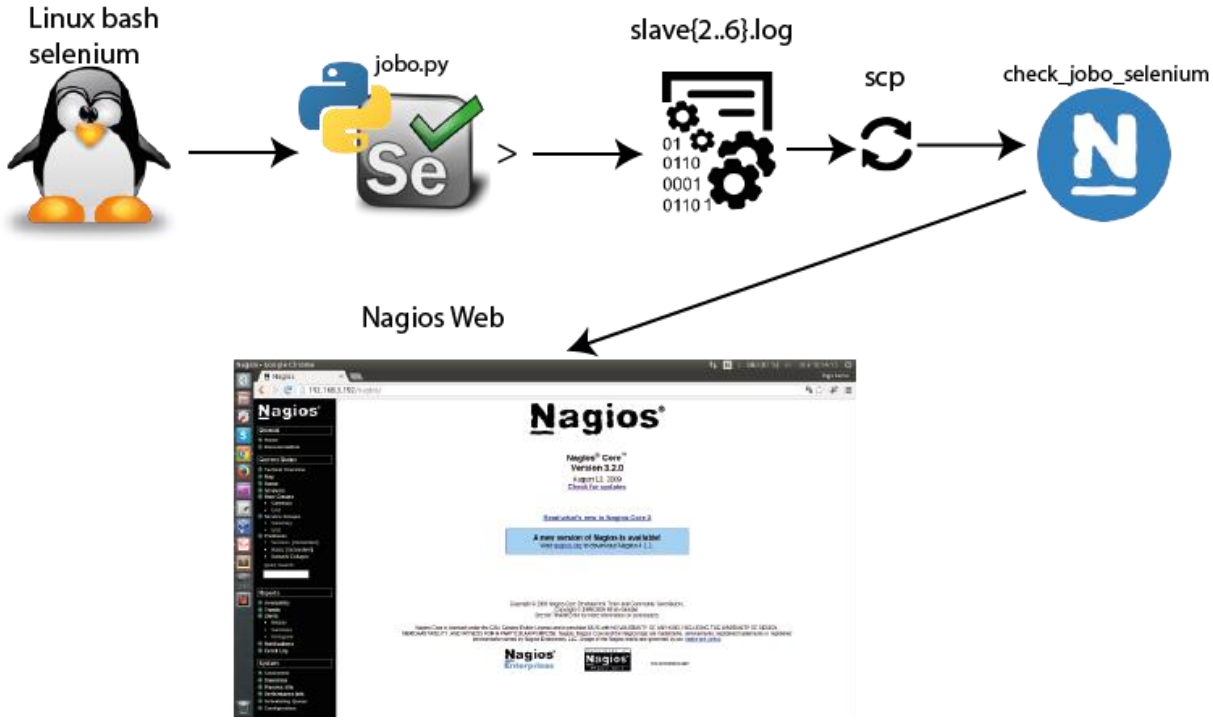

Figura 2.17 Lógica monitoreo Selenium

3.5.3 Programación de rutinas de Selenium

Archivo Selenium:

selenium | jobo.py | check_jobo_selenium

```
#! /bin/bash
while [ 1 -eq 1 ]; do
        for i in {2..6};do
                python jobo.py "http://slave$i.jobomas.com/" > slave$i.log & sleep 30;
        done;
        sleep 300;
        scp slave2.log slave3.log slave4.log slave5.log slave6.log user@IP-Nagios:/tmp/
        killall firefox
done;
```

Figura 2.18 Archivo Selenium

Este archivo con la sentencia while y la comparación 1 igual a 1 se ejecuta de forma infinita. Posteriormente ejecuta un for desde 2 a 6, para poder ejecutar el archivo "jobo.py" escrito en Python, al cual se le pasa como parámetro el url del slave indicado con el contador del for, y todos los resultados de la ejecución se recopilan en un log que lleva el nombre del esclavo en cuestión y se les da un tiempo entre ejecución de 30 segundos para no sobrecargar la máquina. Después de hacer la ejecución sobre los 5 esclavos se hace una copia segura (scp) de todos los log's (2-6) por una conexión ssh al servidor de Nagios en el directorio /tmp. Se pausa de manera explícita el scripy durante 5 minutos (300 segundos), tiempo que se da entre ejecución completa del proceso de monitoreo.Y por último se eliminan todos los procesos del navegador (firefox) y así asegurar que no se queden procesos zombies y la máquina se quede sin recursos para poder seguir ejecutando las rutinas del monitoreo.

Archivo jobo.py

El archivo jobo.py contiene todas las rutinas para hacer el testeo de la funcionalidad, como secuencia de ejecución sigue los siguientes pasos:

selenium | jobo.py | check_jobo_selenium | slave2.log

```
1  #!/usr/bin/python
2  # -*- coding: utf-8 -*-
3  from selenium import webdriver
4  from selenium.webdriver.common.keys import Keys
5  from selenium.webdriver.support import expected_conditions as EC
6  from selenium.webdriver.support.ui import WebDriverWait
7  from selenium.webdriver.common.by import By
8  import datetime
9  import sys
10 url=(str(sys.argv[1]))
11 driver = webdriver.Firefox()
12 driver.maximize_window()
13
14 print "started at ", datetime.datetime.today()
15 try:
16     driver.get(url)
17     driver.find_element_by_link_text("Users").click()
18     print "Home Ok"
19 except:
20     print "Error al obtener home " + url
21     print "finished at ", datetime.datetime.today()
22     driver.close()
23     sys.exit(0)
```

Figura 2.19 Archivo jobo.py

De esta imagen cabe resaltar la importación del webdriver de Selenium, para poder así usar las funciones que Selenium provee. Se importan la utilería para enviar texto a los formularios y demás campos que lo requieran.

Se importan utilerías para acciones de espera ante determinados eventos, y utilerías para buscar elementos en el código HTML de la página a partir de nombres, clases, ID's o bien por una ruta.

Como lo mencione anteriormente este archivo es ejecutado por el archivo Selenium en bash, pasándole el url del esclavo a escanear dicho parámetro en Python se recupera en la variable con la instrucción *url=(str(sys.argv[1]))*. Posteriormente se instancia el navegador a usar (en este caso Firefox por la poca carga en RAM que supone en la ejecución de varias ventanas en comparación con Chrome o Chromiun).

El navegador intenta (try:) recuperar el url que envía el archivo selenium, y esta acción es el primer monitoreo que se ejecuta ya que comprueba que el home o página principal del servidor esté disponible o no, en cualquiera de los casos se manda a escribir el resultado con un *print "información"* en el log. Toda cadena impresa con un print en este archivo se mandara directamente al log del esclavo en cuestión debido a que en el archivo Selenium redirigimos la ejecución del archivo jobo.py a un log.

Es importante mencionar que en caso de tener una excepción en cualquiera de los try's que cada uno comprende un monitoreo; automanticamente se corta la ejecución del archivo con un *sys.exit(0)* debido a que los monitoreos son secuenciales y todos dependen del monitoreo anterior para poder ejecutarse de manera correcta y así liberamos recursos de la máquina además de informar en que sección está el error.

Después de recuperar el url se da click en "Users", que Python lo realiza con la instrucción *driver.find_element_by_link_text("Users").click()* y de manera explícita le decimos al driver que contiene la instancia del navegador que encuentre un link con el texto "Users" y que haga clic sobre él.

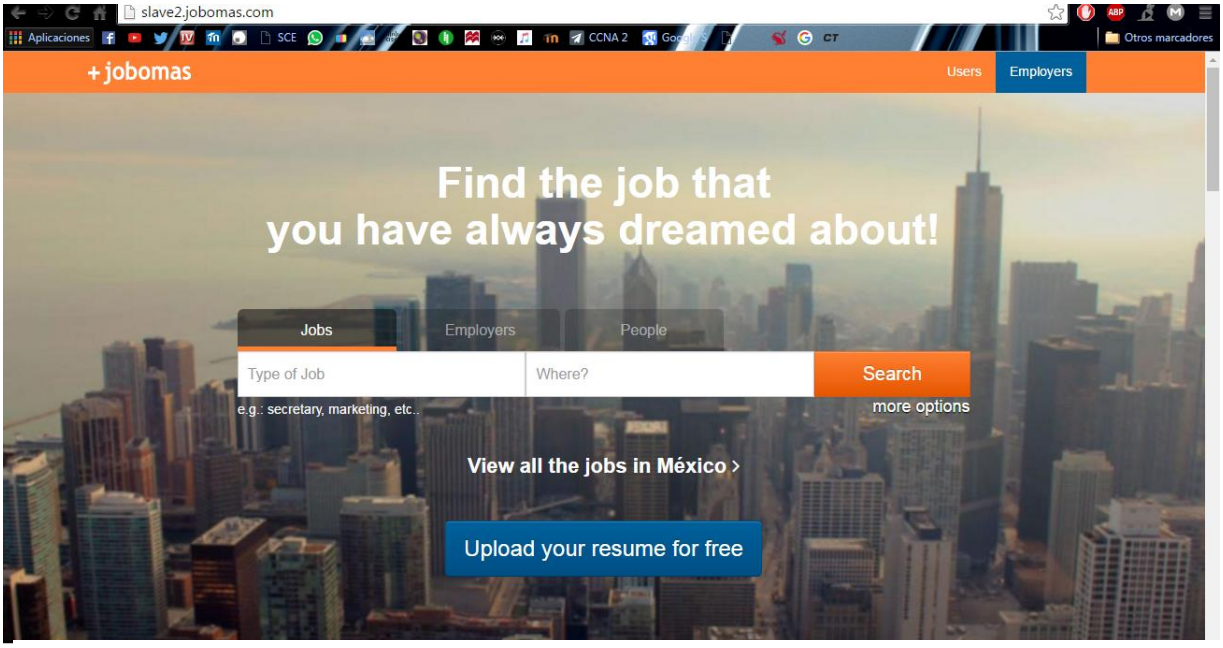

Figura 2.20 Home slave2.jobomas.com

Una vez que Selenium encontró el elemento la página de logueo carga y se muestra, dicha página es la siguiente.

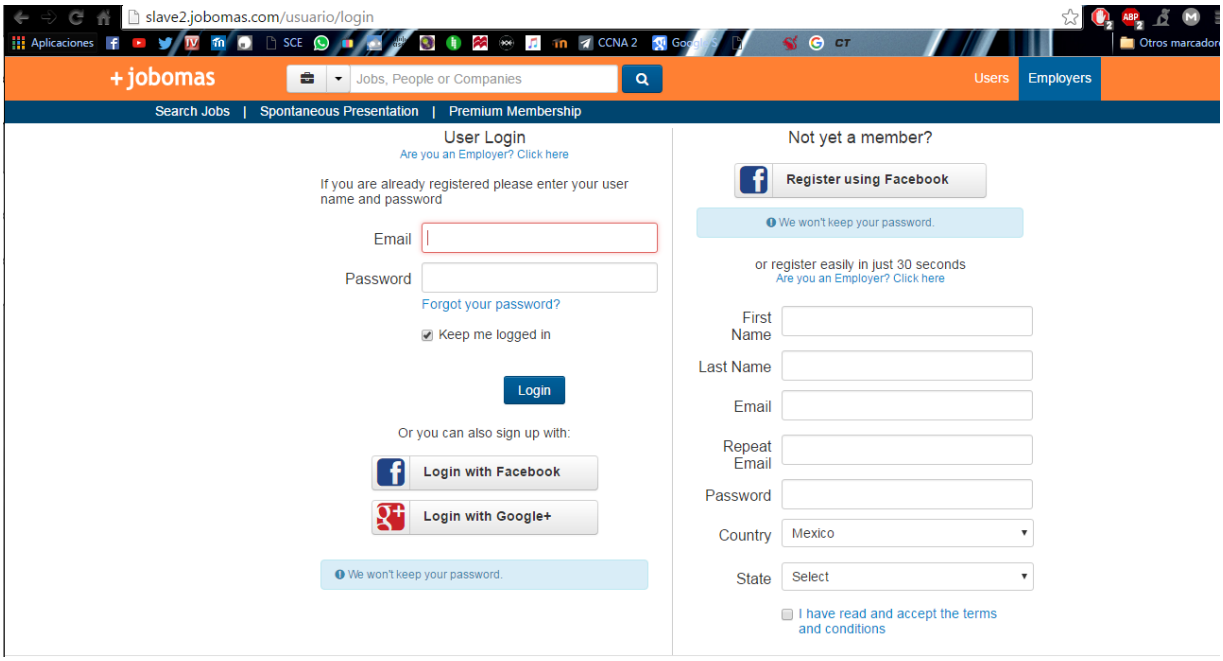

Figura 21 Login jobomas.com

Y se ejecuta el siguiente código:

```
try:
    user = WebDriverWait(driver, 10).until(EC.presence_of_element_located((By.ID, "loginForm_usuario")))
    user.send_keys('user@gmail.com')
    password = WebDriverWait(driver, 10).until(EC.presence_of_element_located((By.ID, "loginForm_password")))
    password.send_keys('password')
    driver.find_element_by_id("loginForm_ingresar").click()
    driver.get(url+"news_feed?utm_source=Inicio_Menu")
    print "Login Ok"
except:
    print "Error: en logeo " + url
    print "finished at ", datetime.datetime.today()
    driver.close()
    sys.exit(0)
```

Figura 2.22 Archivo jobo.py(2)

Este código hace un try de las siguientes instrucciones, primeramente le dice al driver de Selenium a que espere 10 segundos hasta la presencia del elemento localizado por ID llamado “loginForm_usuarios” que es el ID del campo usuario del formulario, se hace de esta manera debido a que Selenium ejecuta las instrucciones muy rápido y a veces tanto que, marca error cuando la página tarda más de 2 segundos en cargar. Una vez localizado el campo de usuario se le manda el correo en modo texto.

De igual manera se localiza el campo de la contraseña llamado “loginForm_password” y se le pasa la contraseña. Ya llenado los campos con la información se procede a localizar el botón de login por el ID del código HTML y se da clic sobre el mismo. Y para finalizar se recupera la página de inicio donde el usuario tiene las noticias de sus contactos y empresas que sigue.

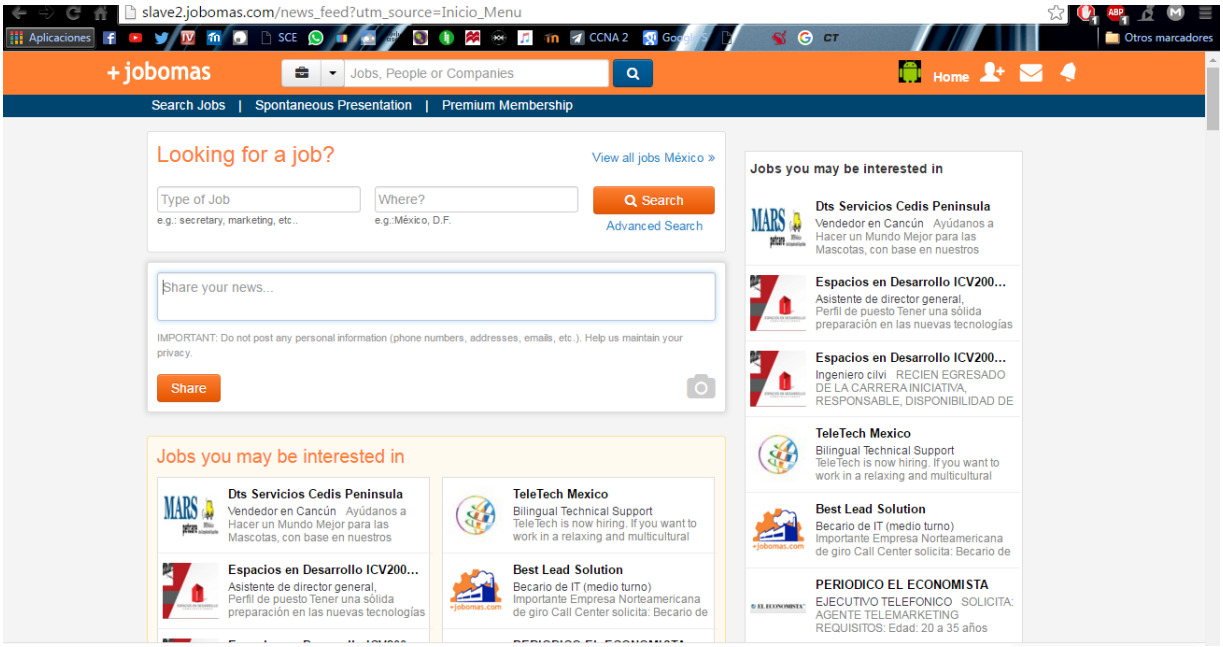

Figura 2.23 Página principal jobomas.com

De existir algún error en cualquiera de las instrucciones pasamos a la excepción donde informamos con un print el url donde está el error o sección además de terminar la ejecución del archivo.

A partir de este punto el monitoreo de la funcionalidad básica de las páginas como lo es la disponibilidad del home, el poder hacer clic sobre elementos y el logueo del usuario está cubierto. Después de comprobar los aspectos anteriores procedemos a monitorear aspectos más complejos de la página como lo son las búsquedas. Las búsquedas pueden ser de 4 tipos: Empleos, Personas, Empresas y del tipo All.

```
39  #Busqueda de empleos
40  try:
41      txtsearch = WebDriverWait(driver, 10).until(EC.presence_of_element_located((By.ID, "buscador-red")))
42      txtsearch.send_keys('Mecanico')
43      WebDriverWait(driver, 10).until(EC.presence_of_element_located((By.ID, "bucador-redsocial"))).click()
44      #cerrar_modal_general
45      WebDriverWait(driver, 10).until(EC.presence_of_element_located((By.ID, "cerrar_modal_general"))).click()
46      driver.implicitly_wait(1)
47      print "Empleos: ", len(driver.find_elements_by_xpath("//p[@class='tituloListado']"))
48  except:
49      print "Error busqueda empleos " + url
50      print "finished at ", datetime.datetime.today()
```

Figura 2.24 Archivo jobo.py(3)

Para realizar este monitoreo primeramente se localiza el campo de texto por su ID “buscador-red” para poder rellenar con la palabra clave a buscar, en este caso Mecánico.

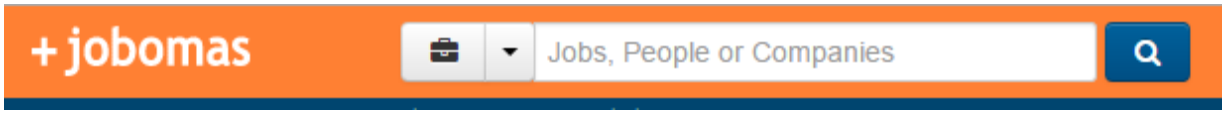

Figura 2.25 Caja de búsqueda jobomas.com

Dicho campo se muestra en la imagen anterior y contiene la leyenda “Jobs, People or Companies” y junto a este el botón para realizar la búsqueda también localizado por su ID “buscador-redsocial”. Una vez hecha la búsqueda se espera a la visibilidad del elemento “modal_general” que es una ventana emergente y se procede a cerrarla con la localización del botón “cerrar_modal_general”, que se muestra en la siguiente imagen.

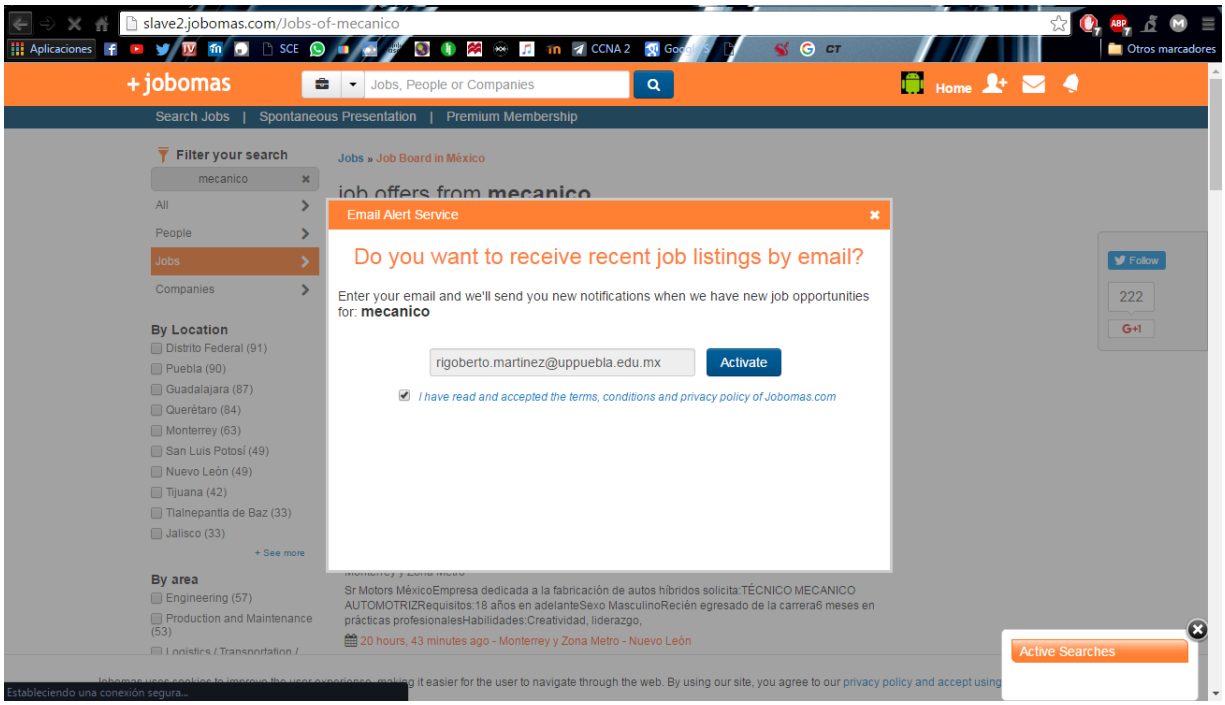

Figura 2.26 Ventana emergente

Ya cerrado la ventana emergente saltan a la vista los resultados de nuestra búsqueda, dichos resultados son contabilizados para cerciorarse de que las búsquedas estén generando contenido como resultado, dichas búsquedas se contabilizan con la instrucción:

- *print "Empleos: ", len(driver.find_elements_by_xpath("//p[@class='tituloListado']"))*

Que hace una búsqueda dentro del condigo HTML de todos los elementos con la etiqueta <p> donde se almacenan los resultados y que la clase de dicho elemento coincida con el nombre “tutiloListado” y los cuenta con la instrucción “len” para por ultimo escribir el resultado en el logo con un print.

Resultado grafico de la búsqueda:

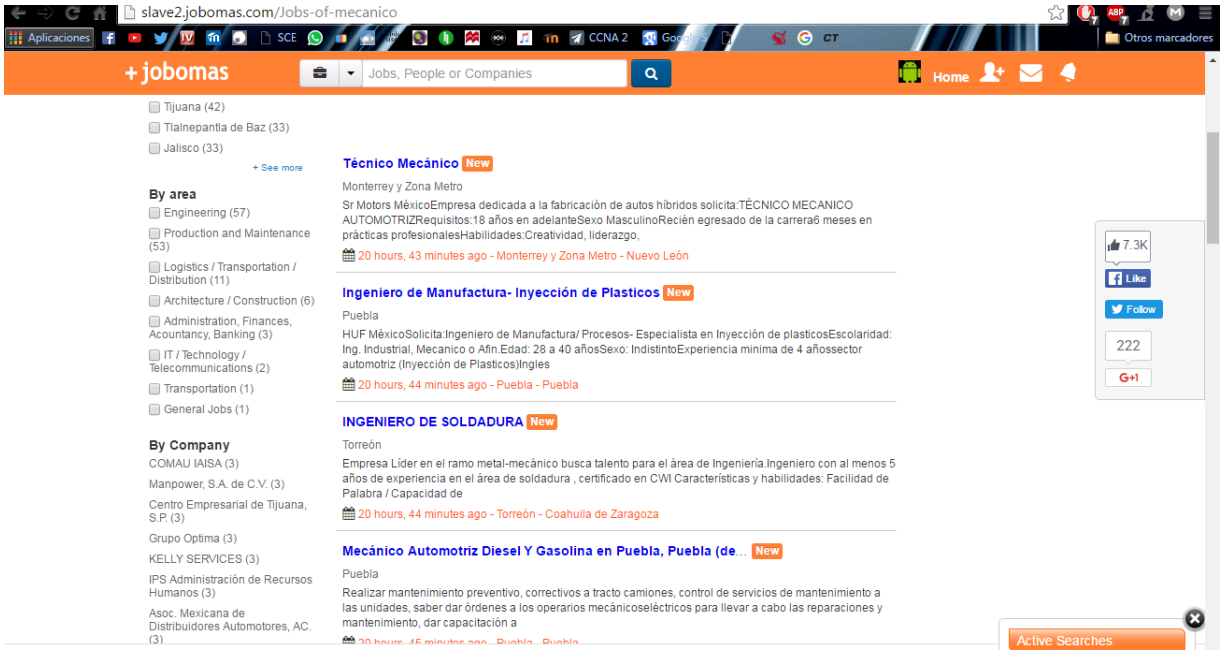

Figura 2.27 Resultado de búsqueda grafica

Resultado de la búsqueda en el código HTML:

```
▼<h2 class="s14" itemprop="title">
  ▼<p class="tituloListado" style="max-width: 435px">
    ▶<a href="/tecnico-mecanico_iid_53894327" title="Técnico Mecánico - Jobs in Mexico" style="color: rgb(51, 51, 51);">…</a>
  </p>
  <span class="label label-warning">New
                                    </span>
```

Figura 2.28 Resultado búsqueda HTM

Como se aprecia el código de un resultado de la búsqueda esta maquetado dentro de la etiqueta <p> y la clase se llama “tituloListado”.

Todas las búsquedas se hacen de esta manera, se abre el menú, se inserta el texto a buscar, se da clic en el botón de búsqueda, se cierra el modal general o ventana emergente, se búsqueda dentro del código las etiquetas correspondientes para poder cerciorarnos de que sean resultados los que contamos y no otros elementos. Sólo cabe resaltar la apertura del menú junto al campo de búsqueda ya que supuso un problema

a la hora de la implementación debido a que no contiene un nombre ni un ID, por lo que se abrió de una manera especial:

- driver.find_element_by_xpath("//button[@class='btn padB8 dropdown-toggle']").click()

Este peculiar elemento se localizó por una ruta, comenzando por los elementos del tipo botón y se usó la clase proveniente del archivo de estilos (CSS) llamada “btn padB8 dropdown-toggle'” y de esta manera se desplego dicho menú.

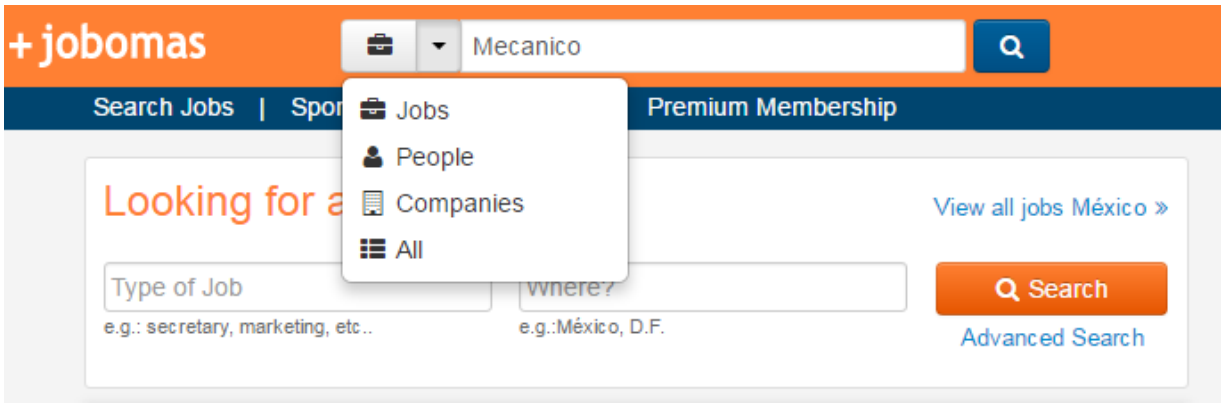

Figura 2.29 Menú de búsqueda

Los monitoreos finales son desplegando el menú del usuario que hace referencia a su perfil, contactos y aspectos personales, de este menú solo se monitorearon aspectos importantes indicados por los dueños de la empresa.

Código para monitoreos finales:

```
99  #Secciones de perfil
100 try:
101     #respuesta del mx
102     print "Abriendo gente que podrias conocer"
103     driver.find_element_by_xpath("//li[@id='menu_perfil']").click()
104     driver.find_element_by_link_text("People you might know").click()
105     driver.implicitly_wait(1)
106     print "Gente que podria conocer mx1: ", len(driver.find_elements_by_xpath("//div[@class='persona']"))
107     #respuesta del slave
108     driver.get(url+"people_you_might_know?utm_source=Red_Gente_Conocer")
109     driver.implicitly_wait(1)
110     print "Gente que podria conocer slave: ", len(driver.find_elements_by_xpath("//div[@class='persona']"))
111 except:
112     print "Error gente que podrias conocer " + url
113     print "finished at ", datetime.datetime.today()
114     driver.close()
115     sys.exit(0)
```

Figura 2.30 Archivo jobo.py(4)

Los monitoreos finales siguen la siguiente lógica:

- se abre el menú.
- se da clic en el link con el texto deseado.
- se busca algún identificador dentro del código HTML.

- se vacía el resultado en el log del esclavo en cuestión.

Y así termina el monitoreo de la funcionalidad de las páginas web de la empresa. Ya para finalizar se indica en el log al hora de termino de ejecución así como se indicó el inicio de la misma *print "finished at ", datetime.datetime.today()* y se cierra el navegador para liberar recursos *driver.close().*

Una vez finalizada la ejecución del archivo jobo.py se copian los log's al servidor de Nagios por una conexión ssh y por el comando scp que lleva la estructura:

- scp [archivo1,2...n] [usuarios@Ip-Dominio:/ruta/destino/]

Los parámetros entre corchetes son intercambiables ya que de igual manera se puede hacer un scp remoto y extraer archivos de un host al que no tengamos acceso de manera física.

3.5.4 Monitoreo de resultados de Selenium con Nagios

Ya echo el copiado de los resultados al servidor de Nagios, ahora le toca a este analizar el resultado y reportar los mismos a su web. El análisis de resultado se lleva a cabo con el archivo “check_jobo_selenium”, este archivo scripteado en bash analiza todos los log’s generados por las ejecuciones de Selenium.

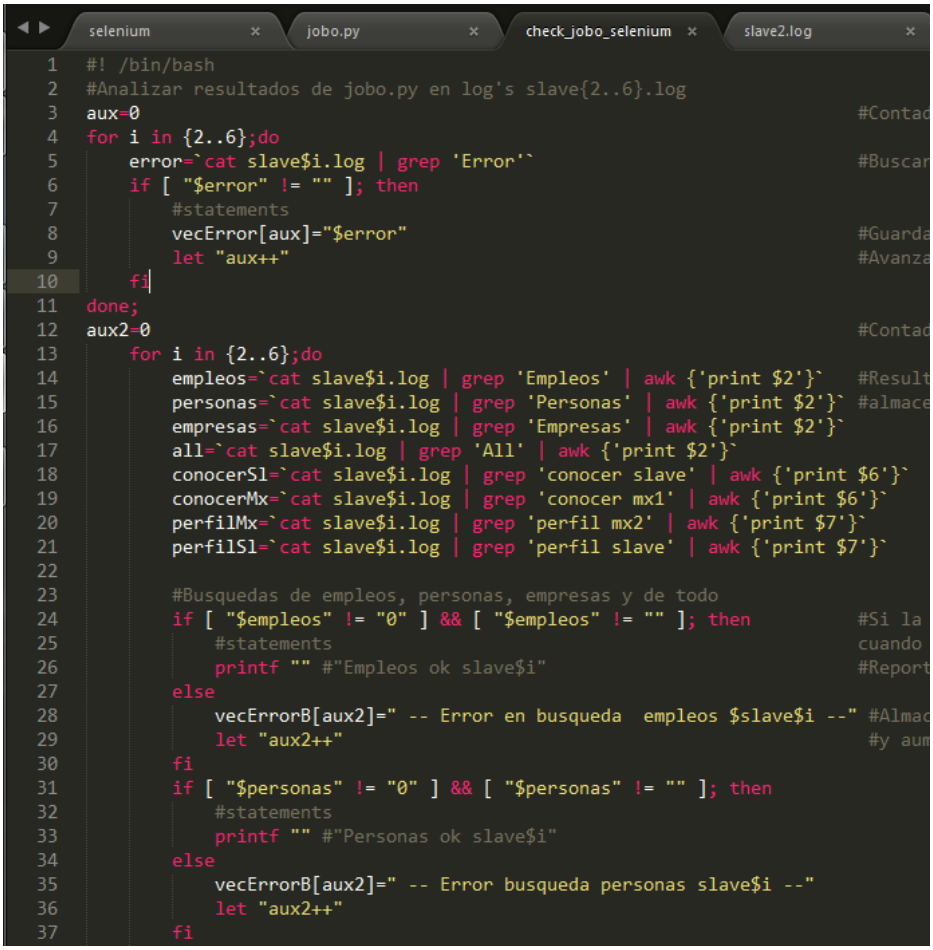

Figura 2.31 Archivo check_jobo_selenium

Se le dio prioridad a la interrupción general de las ejecuciones, debido a que representan un problema muy grande y es lo que se contabiliza a priori. Los errores generales de ejecución se almacenan en el vector “vecError” y los errores de búsquedas se guardan en “vecErrorB”, esto se hizo para brindar de manera simple y a simple vista la información de los errores y en donde se generaron.

Una vez realizado la búsqueda de errores generales y almacenados en el vector “vecError” se sacan los resultados de los monitoreos de las búsquedas y de las secciones. Estos resultados se extrajeron de los log’s de la siguiente manera:

- se recorrió cada log con un for de 2 a 6.
- Se leyó el log con el comando cat

- De la lectura anterior se pasó el resultado por una tubería al comando grep que a su vez busco el texto del resultado de la búsqueda e imprimió la columna dos que es el resultado en numero
- Se comparó el resultado de los comandos anteriores si eran igual a cero o a una cadena vacía, para comprobar para que la búsqueda fuera correcta y que regresara resultados.
- Sí la comparación da como resultado un valor verdadero sólo se imprime una cadena vacía, ya que para el monitoreo sólo nos interesa si las ejecuciones o búsquedas contiene error, de lo contrario se da por hecho que todo está bien en las páginas y para no hacer uso del CPU y de RAM de manera inútil.
- De lo contrario al devolver un valor falso, se almacena en el vector de "vecErrorB" que búsqueda es la que dio el error y en que servidor.

Una vez buscado y almacenado toda la información se procede a deducir el estado del monitoreo:

```
82
83  if [ "${vecError[0]}" == "" ]; then                                    #Sino hay er
84      #statements
85      echo "Todas ejecuciones correctas"
86      if [ $aux2 -eq 0 ]; then                                           #Sino hay er
87          echo "Todas las busquedas correctas"
88          exit 0                                                         #Ok nagios
89      else
90          if [ $aux2 -eq 40 ]
91          then
92              echo "Ninguna busqueda correcta, revisar los logs /tmp/slave{2..6}.log"
93              exit 2
94          else
95              echo "Errores en $aux2 busquedas: ${vecErrorB[*]}"           #Se imprimen
96              exit 1                                                     #Warning nag
97          fi                                                             #de manera c
98      fi
99  else #Si hay errores
100     if [ "${vecError[4]}" != "" ];
101     then                                                               #Si todos lo
102         echo "No se llevo acabo ninguna ejecucion completa"
103         echo "Errores ${vecError[*]}"                                  #Se imprimen
104         if [ $aux2 -eq 40 ]
105         then
106             echo "Ninguna busqueda correcta, revisar los logs /tmp/slave{2..6}.log"
107         else
108             echo "Errores en $aux2 busquedas: \n ${vecErrorB[*]}"      #Se imprimen
109         fi
110         exit 2                                                         #Critical na
111     else
112         echo "Error de ejecucion $aux servidores ${vecError[*]}"       #Se imprimen
113         printf "Busquedas $aux2 incorrectas de 40 \n En: ${vecErrorB[*]}" #Se imprim
114         exit 1
115     fi
116 fi
```

Figura 2.32 Archivo check_jobo_selenium(2)

La deducción de los resultados y el estado de las páginas se programó de la siguiente manera:

- Primero se comparó que el vector "vecError" (que contiene los errores generales) estuviese vacío, de ser verdadero se compara que el valor del contador aux2 usado como índice de "vecErrorB" se encuentre en cero, y de ser así se lanzan a Nagios web los letreros de "Todas las ejecuciones correctas",

“Todas las búsquedas correctas” y se marca el estado del monitoreo en un “Ok”. El motivo de buscar en las comparaciones que a priori todo estuviese correcto es que en los testeos más del 90 por ciento de los resultados fueron un ok total, entonces para no cargar al servidor con procesos innecesarios primero se busca que todo este correcto.

- De devolver un falso la comparación del contador aux2 igual a 0, se compara el mismo contador que sea igual a 40 (total de búsquedas por todas las ejecuciones), de ser así se lanzan a Nagios los letreros: “Todas las ejecuciones correctas”, "Ninguna búsqueda correcta, revisar los logs /tmp/slave{2..6}.log" y se marca el monitoreo como crítico con un exit 2.
- Si la comparación del aux2 igual a 40 da como falso en el else de esta comparación se imprimen cuántos errores de las búsquedas existen y en que servidor esclavo se dieron, enviando a la web de Nagios los letreros: “Todas las ejecuciones correctas”, "Errores en $aux2 búsquedas: ${vecErrorB[*]}". Con el llamado de la variable ${vecErrorB[*]} se imprime todo el contenido del vector. Marcando el monitoreo como warning debido a que solo existen errores en las búsquedas. Y así terminan los posibles escenarios de resultados para la primera comparación.
- En caso de que la comparación de "${vecError[0]}" == "", o sea que el vector de errores generales se encuentre vacío regrese un falso, en el else (sino) se analizan los posibles resultado de este escenario. Como primera instrucción se compara que el vector de errores generales en la posición 4 sea diferente de vacío ("${vecError[4]}" != "") es decir que el vector haya registrado errores generales para todas las ejecuciones. De ser así se compara que el contador del vector de búsquedas sea igual a 40 ($aux2 -eq 40) y de ser verdadero se envían a Nagios los textos "No se llevó a cabo ninguna ejecución completa", "Errores ${vecError[*]}", "Ninguna búsqueda correcta, revisar los log’s /tmp/slave{2..6}.log" siendo este el peor de los escenarios y por ende se marca el monitoreo en crítico con un exit 2.
- De dar como resultado un falso la comparación de $aux2 -eq 40 se imprimen en la web de Nagios los letreros; "No se llevó a cabo ninguna ejecución completa", ”Errores ${vecError[*]}", "Errores en $aux2 búsquedas: \n ${vecErrorB[*]}". Y se marca el monitoreo como critico ya que todos los servidores esclavos tienen errores generales y de búsqueda.
- De no ser verdadera la comparación "${vecError[4]}" != "" indica que hay errores en ciertos servidores y errores en ciertas búsquedas por lo que se imprimen en la web de Nagios los textos: "Error de ejecucion $aux servidores ${vecError[*]}", "Búsquedas $aux2 incorrectas de 40 \n En: ${vecErrorB[*]}" y se marca el monitoreo como en warning.

El marcar en warning o en alerta el monitoreo se debe a que el corte parcial de las ejecuciones o de las búsquedas puede ser resultado de cortes de conexión con internet y no por que en verdad las páginas estén lanzando errores.

Selenium ayuda de una manera muy importante a la empresa a monitorear la funcionalidad de las páginas ya que representa a un usuario virtual, que puede hacer clic, búsquedas, obtener url's y demás acciones que un usuario real lleva a cabo dentro de las mismas. De esta manera con Selenium se automatizan las tareas que un encargado de TI tendría que realizar de manera física perdiendo tiempo en tareas repetitivas y tediosas.

3.6 Lógicas y monitoreos de respaldos.

Para la empresa contratar un servicio de respaldos automáticos con los DataCenters representa un gasto, y además se considera un gasto innecesario debido a que estos respaldos los puede automatizar el departamento de sistemas con simples pero no por eso ineficientes script's en bash.

La lógica de un respaldo como ya se mencionó en el capítulo II, conlleva el análisis de varios factores tanto del lado del servidor donde se encuentra la información como del lado del servidor a donde se va a respaldar la información. No se crearon los respaldos dentro del mismo servidor donde está la información porque de nada serviría tener un respaldo si un servidor es hackeado, atacado o sufre un desperfecto porque tanto la información como el respaldo estan en el mismo servidor, y se perdería todo. Por lo cual los respaldos se sincronizan o copian a servidores dedicados exclusivamente para esto.

Como ya se había mencionado, los principales aspectos a considerar para crear un respaldo es:

- Tamaños de discos duros y espacios disponibles.
- Tamaños de respaldos.
- Horarios de respaldos.
- Días de respaldos.
- Cuántos respaldos se conservan.
- Qué servicios ejecutan los servidores a respaldar.

Todo ello se considera para poder crear un respaldo eficiente y no afectar los servicios en los servidores y brindar la mejor experiencia a los usuarios finales de los proyectos.

Se explicara de manera general el respaldo de una base de datos completa y todo lo que conlleva este respaldo. Aspectos que se consideraron, mejores prácticas para el

respaldo, el porqué de la manera en que se crea el respaldo así como los script's que lo llevan a cabo.

Respaldo de Base de datos X

Antes de realizar el respaldo se llevó a cabo un análisis sobre las variables ya antes descritas, cabe mencionar que es un respaldo local, solo se mueve la información de un disco duro a otro. Información recopilada:

Análisis:

- Peso de la base de datos 75 GB's sin comprimir.
- Uso: Moderación de anuncios y sistemas de gráficas.
- Disco duro libre en el servidor: 300 GB's.
- Horarios de uso de la base de datos: Lunes – Viernes de 7:00 am – 6:00 pm, y de 10:00 pm – 6 00 am (turnos de madrugada de los ejecutivos).
- Base de datos replicada a servidores esclavos.

Información determinada después del análisis.

- Días de respaldo: diario, debido a la importancia y cantidad de la información.
- Horarios de respaldo: diario de 7:00 pm – alrededor de 9:00 pm, variaciones debido al crecimiento de la base de datos. Horario usado para no interrumpir los ejecutivos de la madrugada. Aunque se usó un esclavo para crear el respaldo, en caso de ocurrir un error habría que parar el servidor maestro para ajustar la réplica.
- Peso de la base de datos comprimida: 45 – 50 GB's.
- Se determinó que se conservarían dos respaldos, uno del día anterior y el del día. Debido a la importancia y cantidad de la información.
- Con los programadores responsables del proyecto se decidió que se iba a parar el servidor maestro para ejecutar el respaldo mucho más rápido que si se usara el copiado remoto. Además de que el respaldo se crearía haciendo una copia de la carpeta de la base de datos para posteriormente comprimirla. No se usó hotcopy (herramienta de MySQL) debido a que algunas tablas usan MyISAM como motor y otras InnoDB y debido a ello un respaldo con hotcopy se truncaría.
- También se determinó que el respaldo se monitorearía después de media noche hora de México, para así estar seguro que cualquier error del mismo fuese crítico, y no alertar en falso. Para ello se instalaría un cron para ejecutar el monitoreo del archivo.

Información a monitorear del respaldo:

- Que se creara el archivo del día.

- Que el peso del respaldo no fuera menor a 45 GB’s, debido a que la tendencia de la base de datos es crecer entonces de lo contrario indicaría un fallo o algún problema en la base de datos.
- Que no existiera la carpeta original de la base de datos.
- Que no existieran respaldos de más de un día atrás.
- Estos dos últimos puntos debido a que el consumirían mucho disco duro y podrían llenarlo, dejando sin funcionamiento al servidor además de que el disco duro de destino también aloja más respaldos.

Una vez realizado el análisis y determinado el modo en que se crearía el respaldo además de los puntos críticos que se deberían monitorear se procedió a scriptear el respaldo. Se scriptearon tres archivos, uno que llevar a cabo el respaldo llamado respaldo_dbx, uno para recopilar la información del respaldo llamado info_dbx y el ultimo script para el monitoreo de Nagios llamado monitoreo_dbx.

Respaldo_dbx, que se ejecuta desde un cron a las 7:00 pm todos los días:

Funciones del script:

```
function obtVar()
{
        echo "show slave status\G;" | mysql -u $1 -p$2 | grep Master > /tmp/mysqlStatus
        grep "Read_Master_Log_Pos\|Exec_Master_Log_Pos" /tmp/mysqlStatus | awk '{print $2,$4}'
}
function backup()
{
        if [ $(echo $3 | awk '{print $1}') = $(echo $3 | awk '{print $2}') ];
        then
                echo "Snapshot antes de parar replicacion"
                cat /tmp/mysqlStatus
                echo "parando slave.... $(date)"
                echo "stop slave;" | mysql -u $1 -p$2
                echo "slave detenido $(date)"
                echo "Snapshot despues de parar replicacion"
                echo "show slave status\G;" | mysql -u $1 -p$2 | grep Master
                echo "deteniendo mysql.... $(date)"
                /etc/init.d/mysqld stop
                echo "Mysql detenido  $(date)"
                echo "inicia copiado /var/lib/mysql/DBX .... $(date)"
                cp -r /var/lib/mysql/DBX /mnt/respaldos
                echo "termina copiado de /var/lib/mysql/DBX e inicia Mysql ..."
                /etc/init.d/mysqld start
                echo "Mysql iniciado... iniciando replicacion .... $(date)"
                echo "start slave;" | mysql -u $1 -p$2
                echo "replicacion iniciada ... iniciando crompresion .... $(date)"
                tar -zcvf /mnt/respaldos/DBX$(date +%d-%m-%y).tar.gz /mnt/respaldos/DBX
                echo "compresion exitosa ... inicia borrado de /mnt/respaldos/DBX ... $(date)"
                rm -rf /mnt/respaldos/DBX
                echo "borrado exitoso ... borrando respaldo antiguo (2 dias atras) ... $(date)"
                rm -rf /mnt/respaldos/DBX$(date --date='-2 day' +"%d-%m-%y").tar.gz
                echo "fin script. $(date)"
        else
                echo "Los Log_Pos no son iguales, $(date)"
        fi
}

```

Figura 2.33 Archivo respaldo_dbx

Llamado de funciones:

```
user="user"
password="password"
z=$(backup $user $password "$(obtVar $user $password)")
while [ $(echo $z | grep "Los Log_Pos no son iguales" | wc -l) -gt 0 ] && [ $(date +%H) -lt 22 ];
do
        sleep 600;
        backup $user $password "$(obtVar $user $password)" > /mnt/respaldos/logrespaldoBDX
        z=$(cat /mnt/respaldos/logrespaldoBDX)
done
```

Figura 2.34 Archivo respaldo_dbx(2)

Explicación del script:

- Se definen las variables de conexión a MySQL (user, password).
- En la variable “z” se guardan los resultados de las ejecuciones de la función backup misma que se le pasan: valor1=user, valo2=password, valor3=ejecución de función obtVar que a su vez se le envían los valores de conexión a MySQL.
- Por ende primero se ejecuta obtVar que hace una conexión a MySQL con los parámetros pasado en su llamado ($1 y $2 respectivos a usuario y contraseña), y ejecuta una instrucción dentro de MySQL “show slave status\G;” ya por ultimo hace una búsqueda de la cadena “Master” y lo guarda en un log o archivo de texto llamado mysqlStatus.
- Por último la función obtVar lee el archivo mysqlStatus y busca las cadenas “Read_Master_Log_Pos\|Exec_Master_Log_Pos” que seguidas de ellas contienen los valores de réplica tanto del master como del esclavo. Estos valores determinan el estado de la réplica y si puede o no ser parado tanto el master como el servicio de MySQL. Para proceder estos valores deben ser iguales. Estos dos valores numéricos son el resultado de la ejecución y son pasados como el tercer parámetro a la función backup.
- Ya obtenido los valores de la función obtVar se procede al llamado de backup. Esta función hace una comparación de los valores que arrojo la función obtVar, si son iguales procede a: detener la replicación del esclavo (echo "stop slave;" | mysql -u $1 -p$2), detener MySQL (/etc/init.d/mysqld stop), hacer copia de la carpeta de la base de datos (cp -r /var/lib/mysql/DBX /mnt/respaldos), después de ellos procede a iniciar MySQL (/etc/init.d/mysqld start) y el la replicación del esclavo (echo "start slave;" | mysql -u $1 -p$2), se comprime la carpeta en un archivo .tar.gz y cabe resaltar que al nombre del archivo se le da el nombre de DBX(Fecha de creación).tar.gz donde fecha de creación se define por “date +%d-%m-%y” comando que nos regresa la fecha en formato Día-Mes-Año, borrar la carpeta original “rm -rf /mnt/respaldos/DBX” y por ultimo borrar el respaldo de dos días atrás “rm -rf /mnt/respaldos/DBX$(date --date='-2 day' +"%d-%m-%y").tar.gz”.
- Si los valores de la función obtVar no fuesen iguales la variable “z” contendría la leyenda “Los Log_Pos no son iguales”.
- En el while imprimios la variable “z”, buscamos la leyenda “Los Log_Pos no son iguales”, hacemos el conteo y comparamos si este conteo es mayor a 0. En otras palabras nos cercioramos que el respaldo ya se haya llevado a cabo en la primera ejecución y que la hora sea menor a las 10 (horario de entrada de los ejecutivos de la madrugada). De ser verdad esta comparación se entra en la ejecución del while que lo que hace es esperar 10 minutos, hacer el llamado de las dos funciones, guardar el resultado de las ejecuciones en /mnt/respaldos/logrespaldoBDX y por ultimo vaciar el contenido del archivo en “z”. De lo contrario solo se termina la ejecución del script.

Con este script se crea el respaldo de la base de daos X.

Monitoreo del respaldo.

En la sección de análisis se definieron los aspectos a monitorear del respaldo, todo ello para poder asegurar la integridad del respaldo y de la información contenida en la base de datos X. Así como también ya se mencionó que dos archivos monitorearían el respaldo. Descripción de las rutinas para monitorear el respaldo:

Info_dbx:

```
#! /bin/bash

cd /mnt/respaldos/

if [ -f DBX$(date +%d-%m-%y).tar.gz ] && [ -f DBX$(date --date='-1 day' +"%d-%m-%y").tar.gz  ];
then
    if [[ -d DBX ]]; then
        #statements
            echo "Existe carpeta original"
            exit 2
    else
        pesohoy=$(du -m DBX$(date +%d-%m-%y).tar.gz | awk {'print $1'})
        if [ $pesohoy -gt 45000 ];
        then
            #statements
            echo "Respaldos correctos"
            echo "0" > /tmp/monitoreoRespaldoDBX.log
        else
            echo "Respaldo incorrecto"
            echo "2" > /tmp/monitoreoRespaldoDBX.log
        fi
    fi
else
    echo "Respaldo incorrecto"
    echo "2" > /tmp/monitoreoRespaldoDBX.log
fi
```

Figura 2.35 Archivo info_dbx

Este archivo es el encargado de recabar la información del respaldo y se ejecuta desde un cron 3 horas después de que comienza el respaldo:

- Como primer paso se mueve a la carpeta donde se encuentra el respaldo (cd /mnt/respaldos/).
- Comprueba la existencia del respaldo del día y del día anterior (-f DBX$(date +%d-%m-%y).tar.gz] && [-f DBX$(date --date='-1 day' +"%d-%m-%y").tar.gz). El comando –f comprueba la existencia de un archivo y –d la existencia de un directorio.
- Si la comparación anterior es verdadera, se procede a comprobar la existencia de la carpeta original dentro de la carpeta actual (-d DBX) y si esta es verdadera se manda el letrero “Existe carpeta original” y se marca el monitoreo como crítico. De lo contrario se procede a recopilar la información del peso del archivo

del día en megabytes (pesohoy=$(du -m DBX$(date +%d-%m-%y).tar.gz | awk {'print $1'})) y se guarda en la variable “pesohoy”.

- El siguiente paso es comprobar que el valor de la variable sea mayor a 45000 que equivaldría a alrededor de 45 GB’s.
- Si la comprobación del peso es correcta marcamos el monitoreo como correcto, llenado un log con el valor 0.
- De lo contrario se marca el monitoreo como incorrecto y se envía in 2 al log.
- Y si la comprobación de la existencia de los archivos de los respaldos es incorrecta se marca el monitoreo como critico enviando un 2 al log.

Se envía un 2 o un 0 al log llamado “monitoreoRespaldoDBX.logs” ubicado en /tmp/ debido a que el comando que ejecuta Nagios solo lee este log y dependiendo del valor es la salida que reporta. Se realizó de esta manera debido a que el archivo anterior ejecuta varias instrucciones complejas que supondría un carga en el procesador si se ejecutaran cada minuto, por lo que el archivo anterior solo se ejecuta una vez al día y el monitoreo o el archivo monitore_dbx se ejecuta cada minuto.

Monitoreo_dbx:

```
1  #! /bin/bash
2
3  valor=$(cat /tmp/monitoreoRespaldoDBX.log)
4  cd /mnt/respaldos/
5  echo "Respaldo, DBX$(date +%d-%m-%y): $(du -h DBX$(date +%d-%m-%y).tar.gz | awk {'print $1'})
6                  DBX$(date --date='-1 day' +"%d-%m-%y"): $(du -h DBX$(date --date='-1 day' +"%d-%m-%y").tar.gz | awk {'print $1'})"
7  if [ $valor -eq 0 ];
8  then
9          exit 0
10 else
11         exit 2
12 fi
```

Figura 2.36 2.36 Archivo monitoreo_dbx

Como se mencionó este archivo solo lee el log (valor=$(cat /tmp/monitoreoRespaldoDBX.log)) y lo almacena en una variable “valor”, posterior a ello se mueve a la carpeta donde está el respaldo, imprime el nombre de los archivos y su peso y por ultimo comprueba que “valor” sea igual a 0 ($valor -eq 0), si es correcto se marca el monitoreo como correcto en la web de Nagios o de lo contrario como crítico.

Y por ende como último paso se da de alta el monitoreo dentro de las configuraciones de Nagios, configuraciones vistas al inicio de este capítulo.

Como se puede observar la creación de un respaldo conlleva muchos y variados aspectos que influyen de manera drástica en la manera en que se llevan a cabo. En general la creación de respaldos supone un trabajo extenso y el conocimiento de las herramientas a emplear para ponerlo en funcionamiento.

Conclusión

El monitoreo es una gran herramienta que ayuda a los profesionales de TI a atender y prever problemas en actividades y procesos críticos de negocio, debido a que automatiza la comprobación del correcto funcionamiento de los procesos y además alerta por distintos medios el estado sea cual sea. Además brinda la flexibilidad de poder crear soluciones que los mismos monitoreos lanzan a partir de determinar el estado de los servicios. Además de que en problemas serios reduce las pérdidas a las mismas ya que alerta de manera inmediata cuando las cosas andan mal en los servicios y brinda al personal encargado información eficiente para atender de la mejor manera los problemas.

Considero que el monitoreo es un área muy extensa ya que debido a que, se puede monitorear desde un simple hardware hasta servicios más complejos como bases de datos, elementos de la red como switches, routers, funcionalidad de páginas web en conjunción con otras herramientas. También considero que es un área muy completa en el aspecto profesional porque además de saber configurar las herramientas de monitoreo se tiene que conocer el funcionamiento general de los servicios a monitorear forzando a los administradores de TI a conocer y aprender nuevas tecnologías.

En conclusión se obtuvo el monitoreo eficaz y confiable que se pretendía lograr, debido a bastas pruebas que se realizaron, y confirmaciones cuando se presentaron problemas en los servicios monitoreados debido a que los scripts y monitoreos dieron funcionaron de manera correcta y dieron al personal encargado información relevante sobre el estado de los mismo e indicios de porque había ocurrido el fallo.

Por ultimo quisiera mencionar que ha sido una gran experiencia el haber trabajado con el monitoreo con una herramienta tan flexible y poderosa como lo es Nagios, además de conocer temas y tecnologías nuevas en aspectos como bases de datos, distribución de proceso, servicios en la nube, configuraciones de servidores, administración de servidores, scripteo en bash etc.

Considero mis prácticas profesionales muy satisfactorias y una gran experiencia que me ha abierto el panorama de la gran rama que es la Informática.

Glosario

LATAM: es un concepto étnico-geográfico aparecido en Francia en el siglo XIX para identificar una región del continente americano de habla española y portuguesa como lenguas oficiales o mayoritarias, también incluye los territorios o países de habla francesa.

NRPE: Nagios Remote Plugin Executor, es un agente que aloja el monitoreo de sistemas remotos usando scripts que son alojados en el sistema remoto.

Xinetd: Demonio EXtendido de Internet, es un servicio o demonio que usan gran parte de los sistemas Unix dedicado a administrar la conectividad basada en Internet. xinetd es una extensión más segura del servicio de Internet inetd.

RackSpace: es una compañía de gestión de computación en la nube, fundada en Windcrest, un suburbio de San Antonio, Texas.

DigitalOcean: es un proveedor Estadounidense de servidores virtuales privados, basado en la ciudad de Nueva York. La compañía alquila facilidades de centros de cómputo existentes, incluyendo sitios como Nueva York, Amsterdam, San Francisco, Londres y Singapur.

Ssh: (Secure SHell, en español: intérprete de órdenes seguro) es el nombre de un protocolo y del programa que lo implementa, y sirve para acceder a máquinas remotas a través de una red.

Python: Python es un lenguaje de programación interpretado cuya filosofía hace hincapié en una sintaxis que favorezca un código legible. Se trata de un lenguaje de programación multiparadigma, ya que soporta orientación a objetos, programación imperativa y, en menor medida, programación funcional.

Arpanet: fue una red de computadoras creada por encargo del Departamento de Defensa de los Estados Unidos para utilizarla como medio de comunicación entre las diferentes instituciones académicas y estatales.

Anexos

Código Selenium escrito en bash:

```
#! /bin/bash
#while [ 1 -eq 1 ]; do
                        for i in {2..6};do
              python jobo.py "http://slave$i.jobomas.com/" > slave$i.log & sleep 30;
       done;
       sleep 300;
       scp slave2.log slave3.log slave4.log slave5.log slave6.log root@192.168.3.192:/tmp/
       killall firefox
#done;
```

Codigo jobo.py en Python:

```
#!/usr/bin/python
# -*- coding: utf-8 -*-
from selenium import webdriver
from selenium.webdriver.common.keys import Keys
from selenium.webdriver.support import expected_conditions as EC
from selenium.webdriver.support.ui import WebDriverWait
from selenium.webdriver.common.by import By
import datetime
import sys
url=(str(sys.argv[1]))
driver = webdriver.Firefox()
driver.maximize_window()
print "started at ", datetime.datetime.today()
```

```
try:
   driver.get(url)
   driver.find_element_by_link_text("Users").click()
   print "Home Ok"
except:
   print "Error al obtener home " + url
   print "finished at ", datetime.datetime.today()
   driver.close()
   sys.exit(0)


try:
   user = WebDriverWait(driver, 10).until(EC.presence_of_element_located((By.ID,
"loginForm_usuario")))
   user.send_keys('user@gmail.com')
   password = WebDriverWait(driver, 10).until(EC.presence_of_element_located((By.ID,
"loginForm_password")))
   password.send_keys('password')
   driver.find_element_by_id("loginForm_ingresar").click()
   driver.get(url+"news_feed?utm_source=Inicio_Menu")
   print "Login Ok"
except:
   print "Error: en logeo " + url
   print "finished at ", datetime.datetime.today()
   driver.close()
   sys.exit(0)
   #raise
```

```
#Busqueda de empleos
try:
    txtsearch = WebDriverWait(driver, 10).until(EC.presence_of_element_located((By.ID, "buscador-red")))
    txtsearch.send_keys('Mecanico')
    WebDriverWait(driver, 10).until(EC.presence_of_element_located((By.ID, "bucador-redsocial"))).click()
    #cerrar_modal_general
    WebDriverWait(driver, 10).until(EC.presence_of_element_located((By.ID,
"cerrar_modal_general"))).click()
    driver.implicitly_wait(1)
    print "Empleos: ", len(driver.find_elements_by_xpath("//p[@class='tituloListado']"))
except:
    print "Error busqueda empleos " + url
    print "finished at ", datetime.datetime.today()
    driver.close()
    sys.exit(0)
#Busqueda de personas
try:
    driver.get(url+"news_feed?utm_source=Inicio_Menu")
    WebDriverWait(driver, 10).until(EC.presence_of_element_located((By.ID, "buscador-red")))
    driver.find_element_by_xpath("//button[@class='btn padB8 dropdown-toggle']").click()
    driver.find_element_by_link_text("People").click()
    txtsearch = WebDriverWait(driver, 10).until(EC.presence_of_element_located((By.ID, "buscador-red")))
    txtsearch.send_keys('Mecanico')
    WebDriverWait(driver, 10).until(EC.presence_of_element_located((By.ID, "bucador-redsocial"))).click()
    driver.implicitly_wait(1)
    print "Personas: ", len(driver.find_elements_by_xpath('//article'))
except:
    print "Error busqueda de personas " + url
    print "finished at ", datetime.datetime.today()
    driver.close()
    sys.exit(0)
```

```
#Busqueda empresas
try:
    WebDriverWait(driver, 10).until(EC.presence_of_element_located((By.ID, "buscador-red")))
    driver.find_element_by_xpath("//button[@class='btn padB8 dropdown-toggle']").click()
    driver.find_element_by_link_text("Companies").click()
    txtsearch = WebDriverWait(driver, 10).until(EC.presence_of_element_located((By.ID, "buscador-red")))
    txtsearch.send_keys('Mecanico')
    WebDriverWait(driver, 10).until(EC.presence_of_element_located((By.ID, "bucador-redsocial"))).click()
    driver.implicitly_wait(1)
    print "Empresas: ", len(driver.find_elements_by_xpath('//article'))
except:
    print "Error busqueda de Empresas " + url
    print "finished at ", datetime.datetime.today()
    driver.close()
    sys.exit(0)
#Busqueda All
try:
    WebDriverWait(driver, 10).until(EC.presence_of_element_located((By.ID, "buscador-red")))
    driver.find_element_by_xpath("//button[@class='btn padB8 dropdown-toggle']").click()
    driver.find_element_by_link_text("All").click()
    txtsearch = WebDriverWait(driver, 10).until(EC.presence_of_element_located((By.ID, "buscador-red")))
    txtsearch.send_keys('Mecanico')
    WebDriverWait(driver, 10).until(EC.presence_of_element_located((By.ID, "bucador-redsocial"))).click()
    driver.implicitly_wait(1)
    print "All: ", len(driver.find_elements_by_xpath('//article'))
except:
    print "Error busqueda de All " + url
    print "finished at ", datetime.datetime.today()
    driver.close()
    sys.exit(0)
```

```
#Secciones de perfil
try:
    #respuesta del mx
    print "Abriendo gente que podrias conocer"
    driver.find_element_by_xpath("//li[@id='menu_perfil']").click()
    driver.find_element_by_link_text("People you might know").click()
    driver.implicitly_wait(1)
    print "Gente que podria conocer mx1: ", len(driver.find_elements_by_xpath("//div[@class='persona']"))
    #respuesta del slave
    driver.get(url+"people_you_might_know?utm_source=Red_Gente_Conocer")
    driver.implicitly_wait(1)
    print "Gente que podria conocer slave: ",
len(driver.find_elements_by_xpath("//div[@class='persona']"))
except:
    print "Error gente que podrias conocer " + url
    print "finished at ", datetime.datetime.today()
    driver.close()
    sys.exit(0)
#Who has seen your profile?
try:
    #respuesta mx
    print "Abriendo: quien ha visto tu perfil"
    driver.find_element_by_xpath("//li[@id='menu_perfil']").click()
    driver.find_element_by_link_text("Who has seen your profile?").click()
    driver.implicitly_wait(1)
    print "Quien ha visto tu perfil mx2: ", len(driver.find_elements_by_xpath("//div[@class='well']"))
    #respuesta slave
    driver.get(url+"profile_stats_viewed_by")
    driver.implicitly_wait(1)
    print "Quien ha visto tu perfil slave: ", len(driver.find_elements_by_xpath("//div[@class='well']"))
```

```
except:
   print "Error quien ha visto tu perfil "+ url
   print "finished at ", datetime.datetime.today()
   driver.close()
   sys.exit(0)

print "finished at ", datetime.datetime.today()
driver.close()
```

Código check_jobo_selenium en bash:

```
#! /bin/bash
#Analizar resultados de jobo.py en log's slave{2..6}.log
aux=0
for i in {2..6};do
   error=`cat slave$i.log | grep 'Error'`
   if [ "$error" != "" ]; then
           #statements
           vecError[aux]="$error"
           let "aux++"
   fi
done;
aux2=0
           empleos=`cat slave$i.log | grep 'Empleos' | awk {'print $2'}`
           personas=`cat slave$i.log | grep 'Personas' | awk {'print $2'}`
                   empresas=`cat slave$i.log | grep 'Empresas' | awk {'print $2'}`
                   all=`cat slave$i.log | grep 'All' | awk {'print $2'}`
                   conocerSl=`cat slave$i.log | grep 'conocer slave' | awk {'print $6'}`
              conocerMx=`cat slave$i.log | grep 'conocer mx1' | awk {'print $6'}`
              perfilMx=`cat slave$i.log | grep 'perfil mx2' | awk {'print $7'}`
              perfilSl=`cat slave$i.log | grep 'perfil slave' | awk {'print $7'}`
```

```
#Busquedas de empleos, personas, empresas y de todo
        if [ "$empleos" != "0" ] && [ "$empleos" != "" ]; then
                printf "" #"Empleos ok slave$i"
        else
                vecErrorB[aux2]=" -- Error en busqueda  empleos $slave$i --"
                let "aux2++"
        fi
        if [ "$personas" != "0" ] && [ "$personas" != "" ]; then
                printf "" #"Personas ok slave$i"
        else
                vecErrorB[aux2]=" -- Error busqueda personas slave$i --"
                let "aux2++"
        fi
        if [ "$empresas" != "0" ] && [ "$empresas" != "" ]; then
                printf "" #"Empresas ok slave$i"
        else
                vecErrorB[aux2]=" -- Error busqueda empresas slava$i --"
                let "aux2++"
        fi
        if [ "$all" != "0" ] && [ "$all" != "" ]; then
                printf "" #"All ok slave$i"
        else
                vecErrorB[aux2]=" -- Error busqueda all slave$i --"
                let "aux2++"
        fi
#Busquedas de los perfiles
        if [ "$conocerSl" != "0" ] && [ "$conocerSl" != "" ]; then
                printf "" #"Ok gente conocer slave$i"
        else
                vecErrorB[aux2]=" -- Error busqueda gente podrias conocer slave$i --"
                let "aux2++"
        fi
```

```
if [ "$conocerMx" != "0" ] && [ "$conocerMx" != "" ]; then
                printf "" "Ok gente conocer mx slave$i"
        else
                vecErrorB[aux2]=" -- Error busqueda podrias conocer mx--"
                let "aux2++"
        fi
        if [ "$perfilMx" != "0" ] && [ "$perfilMx" != "" ]; then
                printf "" #"Ok ha visto perfil slave$i"
        else
                vecErrorB[aux2]=" -- Error busqueda ha visto tu perfil mx --"
                let "aux2++"
        fi
        if [ "$perfilSl" != "0" ] && [ "$perfilSl" != "" ]; then
                printf "" #"Ok ha visto perfil slave$i"
        else
                vecErrorB[aux2]=" -- Error busqueda ha visto tu perfil slave$i --"
                let "aux2++"
        fi
    done;
if [ "${vecError[0]}" == "" ]; then
    #statements
    echo "Todas ejecuciones correctas"
    if [ $aux2 -eq 0 ]; then

        echo "Todas las busquedas correctas"
        exit 0

    else
        if [ $aux2 -eq 40 ]
        then
            echo "Ninguna busqueda correcta, revisar los logs /tmp/slave{2..6}.log"
            exit 2
        else
```

```
else
                        echo "Errores en $aux2 busquedas: ${vecErrorB[*]}"
                        exit 1

                fi
        fi
else #Si hay errores
        if [ "${vecError[4]}" != "" ];
        then
                echo "No se llevo acabo ninguna ejecucion completa"
                echo "Errores ${vecError[*]}"
                if [ $aux2 -eq 40 ]
                then
                        echo "Ninguna busqueda correcta, revisar los logs /tmp/slave{2..6}.log"
                else
                        echo "Errores en $aux2 busquedas: \n ${vecErrorB[*]}"
                fi
                exit 2

        else
                echo "Error de ejecucion $aux servidores ${vecError[*]}"
                printf "Busquedas $aux2 incorrectas de 40 \n En: ${vecErrorB[*]}"
                exit 1
        fi
fi
```

Código respaldo_dbx scripteado en bash:

```
#!/bin/bash
function obtVar()
{
	echo "show slave status\G;" | mysql -u $1 -p$2 | grep Master > /tmp/mysqlStatus
	grep "Read_Master_Log_Pos\|Exec_Master_Log_Pos" /tmp/mysqlStatus | awk '{print $2,$4}'
}
function backup()
{
	if [ $(echo $3 | awk '{print $1}') = $(echo $3 | awk '{print $2}') ];
	then
		echo "Snapshot antes de parar replicacion"
		cat /tmp/mysqlStatus
		echo "parando slave.... $(date)"
		echo "stop slave;" | mysql -u $1 -p$2
		echo "slave detenido $(date)"
		echo "Snapshot despues de parar replicacion"
		echo "show slave status\G;" | mysql -u $1 -p$2 | grep Master
		echo "deteniendo mysql.... $(date)"
		/etc/init.d/mysqld stop
		echo "Mysql detenido  $(date)"
		echo "inicia copiado /var/lib/mysql/DBX .... $(date)"
		cp -r /var/lib/mysql/DBX /mnt/respaldos
		echo "termina copiado de /var/lib/mysql/DBX e inicia Mysql ..."
		/etc/init.d/mysqld start
		echo "Mysql iniciado... iniciando replicacion .... $(date)"
		echo "start slave;" | mysql -u $1 -p$2
		echo "replicacion iniciada ... iniciando crompresion .... $(date)"
		tar -zcvf /mnt/respaldos/DBX$(date +%d-%m-%y).tar.gz /mnt/respaldos/DBX
		echo "compresion exitosa ... inicia borrado de /mnt/respaldos/DBX ... $(date)"
		rm -rf /mnt/respaldos/DBX
```

```
echo "borrado exitoso ... borrando respaldo antiguo (2 dias atras) ... $(date)"
        rm -rf /mnt/respaldos/DBX$(date --date='-2 day' +"%d-%m-%y").tar.gz
        echo "fin script. $(date)"
    else
        echo "Los Log_Pos no son iguales, $(date)"
    fi
}
user="user"
password="password"
z=$(backup $user $password "$(obtVar $user $password)")
while [ $(echo $z | grep "Los Log_Pos no son iguales" | wc -l) -gt 0 ] && [ $(date +%H) -lt 22 ];
do
    sleep 600;
    backup $user $password "$(obtVar $user $password)" > /mnt/respaldos/logrespaldoBDX
    z=$(cat /mnt/respaldos/logrespaldoBDX)
done
```

Código información_dbx scripteado en bash:

```
#! /bin/bash
cd /mnt/respaldos/
if [ -f DBX$(date +%d-%m-%y).tar.gz ] && [ -f DBX$(date --date='-1 day' +"%d-%m-%y").tar.gz  ];
then
  if [[ -d DBX ]]; then
    #statements
      echo "Existe carpeta original"
      echo "2" > /tmp/monitoreoRespaldoDBX.logs
  else
    pesohoy=$(du -m DBX$(date +%d-%m-%y).tar.gz | awk {'print $1'})
    if [ $pesohoy -gt 45000 ];
    then
```

```
#statements
        echo "Respaldos correctos"
        echo "0" > /tmp/monitoreoRespaldoDBX.log
    else
        echo "Respaldo incorrecto"
        echo "2" > /tmp/monitoreoRespaldoDBX.log
    fi
  fi
else
  echo "Respaldo incorrecto"
  echo "2" > /tmp/monitoreoRespaldoDBX.log
fi
```

codigo monitoreo_dbx script en bash:

```
#! /bin/bash
valor=$(cat /tmp/monitoreoRespaldoDBX.log)
cd /mnt/respaldos/
echo "Respaldo, DBX$(date +%d-%m-%y): $(du -h DBX$(date +%d-%m-%y).tar.gz | awk {'print $1'})
                DBX$(date --date='-1 day' +"%d-%m-%y"): $(du -h DBX$(date --date='-1 day' +"%d-%m-%y").tar.gz |
awk {'print $1'})"
if [ $valor -eq 0 ];
then
    exit 0
else
    exit 2
fi
```

Bibliografía

[1] Configuración Nagios en Centos

Augusto Sepulveda. (2009). Configuración de Nagios en Centos. Febero 2014, de NZXT Telecomunicaciones de México SA de CV Sitio web: http://documents.mx/documents/nagios.html

[2] Linux Máxima Seguridad

Anonimo. (Enero 12, 2002). Linux Máxima Seguridad. Madrid: Prentice Hall.

[3] Iptables

ArchWiki. (Abril 2016). Iptables. Abril 2016, de Archlinux Sitio web: https://wiki.archlinux.org/index.php/Iptables_(Espa%C3%B1ol)

[4] Selenium Python Bindings

Baiju Muthukadan. (2014). Selenium Python Bindings. Abril 2016, de Baiju Muthukadan Sitio web: http://selenium-python.readthedocs.org/installation.html

[5] Tesis Monitoreo del comportamiento de servidores de aplicaciones

Julian Selley. (Abril 2008). Monitoreo del comportamiento de servidores de aplicaciones. Febrerp 2016, de Instituto Politécnico Nacional Sitio web: http://www.saber.cic.ipn.mx/cake/SABERsvn/trunk/Repositorios/webVerArchivo/305/1