

Análisis y monitorización de aplicaciones a través de plugins con Naemon

Sofía Fernández Moreno

Universidad de Granada

Septiembre de 2019



UGR

Universidad
de Granada



Objetivos del proyecto

Estado del arte

Realización del despliegue de Naemon

Pruebas de carga

Pruebas de carga en un sistema

Modelado de las pruebas

Conclusiones y futuros trabajos

Objetivos del proyecto

iones y futuros trabajos

Objetivos del proyecto

Estado del arte

Realización del despliegue de Naemon

Pruebas de carga

Pruebas de carga en un sistema

Modelado de las pruebas

Conclusiones y futuros trabajos

¿Qué es la monitorización?

Comparativa herramientas de monitorización

Naemon

Estado del arte

¿Qué es la monitorización?

La **monitorización** consiste en la manera de tomar las medidas preventivas y consecuentes con la información que se obtienen de todos los dispositivos que se encuentran conectados a una red, para evitar posibles eventos que hacen que se interrumpan el correcto funcionamiento de alguno de ellos

Dentro de este concepto es importante aplicar el uso del protocolo **SNMP** (Simple Network Management Protocol).

***SNMP** permite el intercambio de información amplia entre los diferentes dispositivos de red mediante consultas de forma remota (polling) y mediante mensajes basándose en eventos (traps).*

Objetivos del proyecto

Estado del arte

Realización del despliegue de Naemon

Pruebas de carga

Pruebas de carga en un sistema

Modelado de las pruebas

Conclusiones y futuros trabajos

¿Qué es la monitorización?

Comparativa herramientas de monitorización
Naemon

Comparativa herramientas de monitorización



Nagios®

N
naemon

Estado del arte

Realización del despliegue de Naemon

Pruebas de carga

Pruebas de carga en un sistema

Modelado de las pruebas

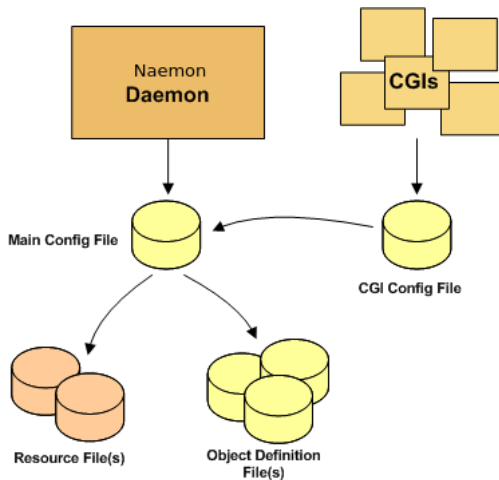
Conclusiones y futuros trabajos

¿Qué es la monitorización?

Comparativa herramientas de monitorización

Naemon

Archivo de configuración principal



Tipos de objetos

*La **configuración** de los distintos equipos y servicios a monitorizar se puede realizar a través de la carpeta **/etc/naemon/conf.d***

Podemos encontrar como objetos de definición:

- Hosts
- Servicios
- Comandos
- Contactos
- Periodos de tiempo

Estado del arte

Realización del despliegue de Naemon

Pruebas de carga

Pruebas de carga en un sistema

Modelado de las pruebas

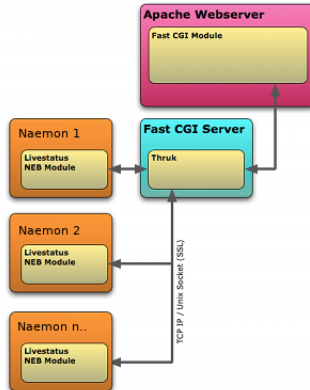
Conclusiones y futuros trabajos

¿Qué es la monitorización?

Comparativa herramientas de monitorización

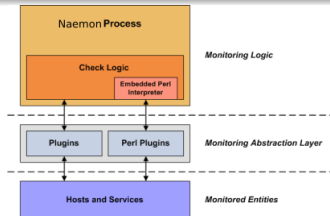
Naemon

Thruk



Uso de plugins

Los **plugins** actúan como una capa de abstracción entre la lógica de supervisión presente en el demonio de Naemon y los servicios y hosts reales que se están supervisando.



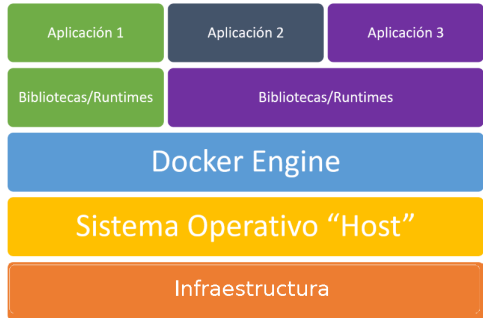
El inconveniente de usar plugins en Naemon es que éste no tiene reflejo de qué es lo que está monitorizando, simplemente rastrea los cambios en el estado de esos recursos.

Realización del despliegue de Naemon

Objetivos del proyecto
Estado del arte
Realización del despliegue de Naemon
Pruebas de carga
Pruebas de carga en un sistema
Modelado de las pruebas
Conclusiones y futuros trabajos

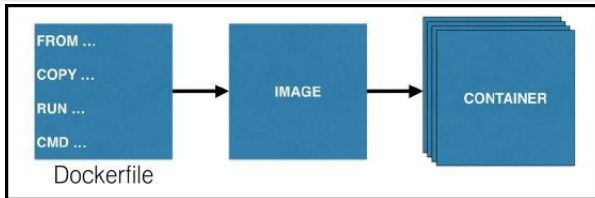
Entorno de desarrollo
Desarrollo de despliegue

Uso de contenedores



Creación de archivo Dockerfile

*Para poder realizar el despliegue tendremos que crear el fichero **Dockerfile**. Con este fichero construiremos la imagen de Naemon de forma automática, leyendo las instrucciones que le indiquemos.*



Archivo de ejecución ENTRYPOINT (run.bash)

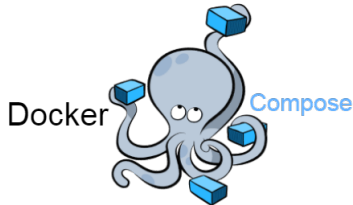
```
WEB_USERS_FULL_ACCESS=${WEB_USERS_FULL_ACCESS:-false}
if [ $WEB_USERS_FULL_ACCESS == true ]
then
    sed -i 's/authorized_for_\(.\\+\)=thrukadmin/authorized_for_\1=*/'
    /etc/thruk/cgi.cfg
fi
```

Orquestación de las aplicaciones

La **orquestación** aplicada en este despliegue consiste en que el sistema requiere una configuración más manual de los recursos y no permite el escalado de forma muy eficiente.

Docker-Compose

Permite la orquestación estática orientada a un funcionamiento más centrado en un solo servidor, esta nos permite a través de un fichero YML, la definición y ejecución de aplicaciones Docker en múltiples contenedores.



Pruebas de carga

Definición

La **prueba de carga** se trata de una prueba que generalmente observa el comportamiento de una aplicación bajo una serie de peticiones.

¿Sólo monitorizar?

Para este proyecto se ha elegido la opción de realizar **pruebas de carga** para poder simular el uso de concurrencia real en nuestro despliegue de Naemon. Además usaremos este tipo de prueba puesto que queremos medir la *performance* o rendimiento de un sitio web, el cual mencionaremos más adelante y este tipo de prueba nos será de gran utilidad.

Objetivos del proyecto
Estado del arte
Realización del despliegue de Naemon
Pruebas de carga
Pruebas de carga en un sistema
Modelado de las pruebas
Conclusiones y futuros trabajos

Comparativa de herramientas
Locust

Comparativa de herramientas



Funcionamiento de Locust

Debemos activar el servidor web y luego para poder usar Locust debemos configurar un archivo **locustfile.py** que contará con el código necesario para realizar las pruebas de carga necesarias, distribuyendo la prueba de rendimiento en diferentes máquinas para que se pueda crear más carga en la aplicación.

Locustfile

- **TaskSet:** se trata de una colección de tareas, es decir, define el comportamiento del usuario. Para cada acción definida debe definirse la anotación @task.
- **HttpLocust:** utilizada para cargar la prueba de rendimiento de un sistema en el servidor. Representa un usuario el cual será atacado y que será probado con carga. Dicha clase crea un atributo cliente que se trata del cliente HTTP que se encarga de realizar el enlace entre la sesión y las peticiones.

```
from locust import HttpLocust, TaskSet, task
```

```
from locust import HttpLocust, TaskSet, task

def login(l):
    l.client.post("/wp-login.php", {"username": "naemon", "password":
    "naemon"})

def logout(l):
    l.client.post("/wp-login.php?loggedout=true", {"username": "naemon",
    "password": "naemon"})

class UserBehavior(TaskSet):
    def on_start(self):
        login(self)

    def on_stop(self):
        logout(self)

    @task(2)
    def root(self):
        self.client.get('/')

    @task(1)
    def host(self):
        self.client.get('/?p=1')

class WebsiteUser(HttpLocust):
    task_set = UserBehavior
    min_wait = 5000
    max_wait = 9000
```

Pruebas de carga en un sistema

- Objetivos del proyecto
- Estado del arte
- Realización del despliegue de Naemon
- Pruebas de carga
- Pruebas de carga en un sistema**
- Modelado de las pruebas
- Conclusiones y futuros trabajos

Estado del arte

Realización del despliegue de Naemon

Pruebas de carga

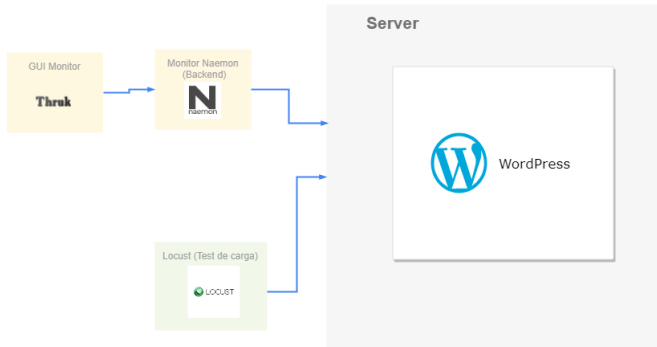
Pruebas de carga en un sistema

Modelado de las pruebas

Conclusiones y futuros trabajos

Creación de archivo docker-compose

Estructura Docker-Compose



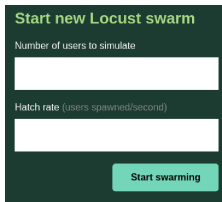
Enlazado con Locust

Para poder enlazar Locust con el sistema debemos adaptar la configuración interna del archivo **locust.config.json**. Este archivo será añadido al fichero **docker-compose.yml** junto al **locustfile.py**. Dicho archivo nos servirá para especificar la URL raíz de la API a la que se va a redirigir la API de Locust, junto con la lista de nombres de clase para las subclases de Locust que se usarán en la prueba.

```
{  
  "target": "http://wordpress",  
  "locusts": ["WebsiteUser"]  
}
```

Interfaz de Locust









- **Number of users simulate:** serán los números de usuarios que queremos asignar a la ejecución de las pruebas
- **Hatch rate:** representa por cada segundo, cuántos usuarios se agregarán a los usuarios actuales hasta la cantidad total de usuarios. Por cada hatch realizado Locust llama a la función `on_start` si existe.



The screenshot shows a dark-themed web interface for starting a new Locust swarm. It features two input fields: 'Number of users to simulate' and 'Hatch rate (users spawned/second)'. A green button labeled 'Start swarming' is positioned at the bottom right of the form.

Creación de host y servicios en Naemon

La carpeta `/etc/naemon/conf.d/` cuenta con el siguiente contenido:

Nombre	1
 templates/	
 commands.cfg	
 contacts.cfg	
 localhost.cfg	
 printer.cfg	
 switch.cfg	
 timeperiods.cfg	
 windows.cfg	

Dentro de la carpeta `templates` se modificarán los archivos `host.cfg`, `services.cfg` para realizar las pruebas de rendimiento del sistema implementado en Naemon.

Creación de host y servicios en Naemon II

Añadiremos en el archivo **DockerFile** creado anteriormente las siguientes líneas:

```
RUN rm -rf /etc/naemon/conf.d/localhost.cfg  
ADD definition/wordpresshosts.cfg /etc/naemon/conf.d/wordpresshosts.cfg  
ADD  
definition/wordpressservices.cfg /etc/naemon/conf.d/wordpressservices.cf  
g
```

Modelado de las pruebas

¿Qué es la carga de trabajo?

La **carga de trabajo** es el conjunto de todas las peticiones que el sistema recibe de su entorno durante un periodo de tiempo dado.

El análisis de la carga es un papel fundamental en cualquier estudio en los que hay que determinar **índices de rendimiento**, estos se encuentran directamente relacionados con la carga y no se pueden expresar de forma independiente a ésta. Además el índice de rendimiento siempre debe ir determinado de la información de la carga bajo la que fue determinado.

PNP4Nagios en Dockerfile

PNP4Nagios es un modulo para Naemon y además para Nagios que analiza los datos de rendimiento de los servicios que tengamos implementados en cada host, almacena automáticamente los datos en bases de datos **RRD** (bases de datos Round Robin).



Exportación de datos en CSV

Para ello la API nos ofrece la forma de exportación a CSV de la siguiente forma:

`/pnp4nagios/xport/ < format >?host =< hostname > &srv =< servicedesc >`

- `< format >` especifica el formato de exportación, teniéndose la opción de XML, JSON y CSV.
- `< hostname >` especificaremos el nombre del host, en este caso introducimos el nombre wordpress.
- `< servicedesc >` especificaremos el servicio que queremos exportar su información.

Resultados



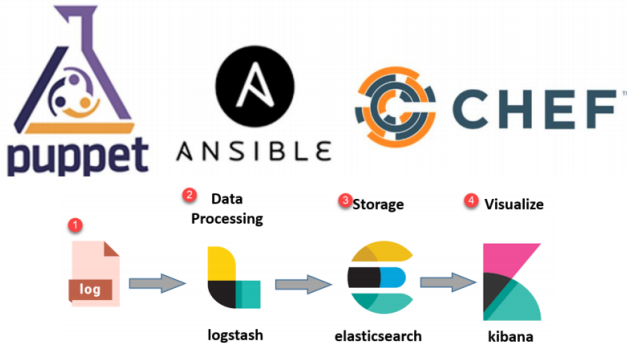
Conclusiones y futuros trabajos

Conclusiones

Objetivos del proyecto
Estado del arte
Realización del despliegue de Naemon
Pruebas de carga
Pruebas de carga en un sistema
Modelado de las pruebas
Conclusiones y futuros trabajos

Conclusiones
Trabajos futuros

Trabajos futuros



Referencias



DevOps, Concept of DevOps,
<https://azure.microsoft.com/es-es/overview/what-is-devops/>.



Raygun 2019, DevOps Tools,
<https://raygun.com/blog/best-devops-tools/>.

Gracias por su atención



sofiafernandezmoreno/TFG