Universidad Nacional de Quilmes

Configuración de un servicio de transformación de información no estructurada a formato JSON sobre un servidor Debian

Ariel Alvarez

Universidad Nacional de Quilmes, Laboratorio de Sistemas Operativos y Redes

10 de diciembre de 2015

Este trabajo busca conformar una guía paso a paso de cómo configurar un servicio en `NodeJS` que se ejecute de manera periódica sobre un servidor `Debian`, y analice determinados sitios web extrayendo información y generando archivos con formato `JSON` a partir de los datos procesados.

1. Introducción

La mayor parte de la información en internet se encuentra expuesta en formatos no estructurados, es decir, dentro de código HTML sin una estructura genérica o estandarizada. Cuando ciertos sistemas requieren acceder a esta información, es necesario primero organizarla en una estructura más sencilla de manejar.

Para esto usaremos un Scrapper, una herramienta que se encarga de procesar sitios a partir de patrones predefinidos, extrayendo información y guardándola en un formato estándar.

En este caso, se utilizará un script escrito en Javascript que será ejecutado periódicamente por NodeJS. El resultado de ese script es un archivo con formato JSON que se guardará en un directorio compartido con una instancia de un Docker corriendo Nginx, la cual servirá el arhcivo por su puerto 80.

De esta manera, el principal acceso público queda aislado en una instancia de docker con la última versión de Nginx, eliminando varios posibles problemas de seguridad.

2. Configuración del Entorno

2.1. Debian

Como sistema operativo host se va a usar **Debian Jessie**, pero puede extrapolarse a cualquier sistema operativo que soporte *NodeJS* y *Docker* .

2.2. NodeJS

Instalamos NodeJS:

```
curl -sL https://deb.nodesource.com/setup_0.12 | sudo bash -
sudo apt-get install -y nodejs
```

Cabe aclarar que, siendo un script que se descarga desde el exterior, es siempre recomendable leerlo antes de ejecutarlo.

2.3. Paquete X-Ray para NodeJS

La librería que utilizaremos para programar el Scrapper se llama X-Ray. En lugar de isntalarla globalmente, la agregaremos como dependencia a nuestro proyecto. Para esto creamos un archivo **package.json** y agregamos las dependencias necesarias:

```
{
  "name": "eardrop",
  "version": "0.0.1",
  "dependencies": {
    "glob": "^6.0.1",
    "gm": "^1.21.1",
    "libxmljs": "0.15.0",
    "lodash": "^3.10.1",
    "needle": "0.10.0",
    "request": "^2.67.0",
    "superagent-proxy": "^1.0.0",
    "x-ray": "^2.0.2"
  }
}
```

Ademaś de *X-Ray*, tenemos otras dependencias como: *superagent-proxy* nos permite configurar proxys para las requests que realicemos (en caso de que queramos evitar que la IP de nuestro servidor de scrapping sea bloqueada al hacer muchos requests consecutivos). *request* nos permite reqlizar requests HTTP planas, con lo cual resulta útil para descargar contenidos tales como imágenes.

Para instalarlas, debemos correr el siguiente comando dentro del directorio donde se encuentra **package.json**:

```
npm install
```

Esto instalará las dependencias de manera local al proyecto, evitando posbiles problemas de versiones con librerías globales.

2.4. Docker

Instalamos Docker e iniciamos el daemon:

```
sudo apt-get install docker-engine
sudo service docker start
```

3. Programación del scrapper

En este ejemplo vamos a conseguir un listado de las noticias destacadas que aparecen en el portal de la UNQ:

```
x('http://www.unq.edu.ar/noticias/', 'aside#listado',{
    noticias:x('li', [
        {
            date:'div.fecha',
            title:'h3 a',
            description:'div.bajada a',
            thumbnail:'figure a img@src'
        }
    ])
}).write('../out/posts.json');
```

X-Ray hace un request a **http://www.unq.edu.ar/noticias/** y se trae el HTML correspondiente a dicha URL. Luego lo recorre utilizando las reglas explicitadas: obtener el contenido del tag *aside* cuyo id es *listado* y luego, por cada elemento *li* que encuentre, crear objetos noticia rellenando sus atributos.

El código completo, que incluye imports para descargar imágenes, transformar su formato, utilizar proxy, etc, es:

```
var
    Xray = require('x-ray'),
    proxy = require('./proxy-driver'),
    fs = require('fs'),
    glob = require("glob"),
    request = require('request'),
    gm = require('gm').subClass({imageMagick: true}),
    _ = require('lodash');

var x = Xray();

x.delay(300, 500);
x.throttle(1, 200);
//x.driver(proxy());

x('http://www.unq.edu.ar/noticias/', 'aside#listado',{
    noticias:x('li', [
        {
            date:'div.fecha',
            title:'h3 a',
            description:'div.bajada a',
            thumbnail:'figure a img@src'
        }
    ])
}).write('../out/posts.json');
```

3.1. Generando un archivo JSON

Nótese que la última línea de nuestro script escribe los resultados en un archivo **posts.json** dentro del directorio **out**.

Este directorio será luego compartido con la instancia de docker, que será la encargada de servirlo por HTTP.

3.2. Usando proxy

Para utilizar un proxy, es posible extender el cliente HTTP que usa el script para soportar un driver customizado, por ejemplo:

```
var superagent = require('superagent');

// extend with Request#proxy()
require('superagent-proxy')(superagent);
/**
 * Export `driver`
 */

module.exports = driver;

/**
 * Custom proxy HTTP driver
 *
 * @param {Object} opts
 * @return {Function}
 */

function driver(opts) {
    var agent = superagent.agent(opts || {});
    var proxy = 'http://104.42.106.203:8080';

    return function http_driver(ctx, fn) {
        agent
            .get(ctx.url)
            .proxy(proxy)
            .set(ctx.headers)
            .end(function(err, res) {
                if (err && !err.status) return fn(err);

                ctx.status = res.status;
                ctx.set(res.headers);

                ctx.body = 'application/json' == ctx.type
                    ? res.body
                    : res.text;
```

```
                // update the URL if there were redirects
                ctx.url = res.redirects.length
                    ? res.redirects.pop()
                    : ctx.url;

                return fn(null, ctx)
            })
    }
}
```

Nos permite inyectar un proxy público, que en el ejemplo se encuentra harcodeado con la IP *104.42.106.203*

4. Servidor HTTP estático usando Nginx dockerizado

Para servir el archivo JSON generado, vamos a utilizar una instancia de docker conteniendo un servidor Nginx. Al correr la instancia, se le monta un directorio del sistema Host, de tal manera que pueda interactuar con archivos generados externamente.

Primero obtenemos la imagen con Nginx:

```
docker pull nginx
```

Luego si lo que queremos hacer es únicamente servir contenido estático, como es nuestro caso, alcanza con correr el contenedor usando:

```
docker run
          --name static-http-server
          -v ./out/:/usr/share/nginx/html:ro
          -d nginx
```

Para evitar escribir manualmente este comando cada vez que se quiera iniciar la instancia del servidor, creamos un archivo **start.sh** que lo contenga, y le damos permiso de ejecución

4.1. Configurando Cron jobs

Dependiendo la la velocidad de cambio del contenido que se desee escrappear, y la necesidad de contenido actualizado procesado, pueden configurarse distintas frecuencias. En nuestro ejemplo vamos a usar una frecuencia diaria. El primer paso es escribir un script que ejecute nuestro programa, alcanza con:

```
#!/bin/bash
node /path/to/script/eardrop.js
```

Llamado **scrapper**, dándole los permisos de ejecución apropiados. Este archivo luego se mueve al directorio de sistema /**etc**/**cron.daily**. Diariamente, el daemon Cron ejecuta todos los scripts que encuentra en este directorio.

5. Conclusión

Vimos entonces cómo es posible ejecutar de manera periódica un script que tome información de sitios y la guarde en archivos json, y cómo estos archivos JSON pueden ser servidos utilizando un servidor HTTP. Para simplificar la configuración del servidor, se hizo uso de Docker, el cual nos permitió tener un entorno actualizado y fácilmente actualizable con un servidor Nginx instalado; de manera tal que los usuarios del servicio accederán al sistema dentro del contenedor, reduciendo riesgos de seguridad.

Una mejora posible a esta implementación es encapsular el ambiente NodeJS también dentro de un docker, y utilizar el directorio compartido como usa suerte de pipe entre

ambos. Esto implicaría que la configuración del CRON también puede ser versionable dentro de Docker, sin impactar en la configuración del sistema Host.