Monitoratge de servidors a temps real

Aplicacions i Serveis d'Internet

Erik C.

Aleix G.

Toni S.

Oscar V.

May 24, 2018

Continguts

Introducció **2**

Python **3**

PHP **6**

Laravel, PHP i MySQL **6**
- User . . . 6
- Server . . . 7
- Net, Pid, Disk, CPU, Mem, Net . . . 9
- Address . . . 9
- Controller . . . 10
- Routes: . . . 12

Web socket **12**

Interfície **13**

Autentificació **13**

Llista de Figures

1 Esquema moduls . . . 6
2 Interfície web . . . 13

Llista de Codis

1 installer . . . 2
2 monitoring.py . . . 3
3 User.php . . . 6
4 Server.php . . . 7
5 Disk.php . . . 9
6 Address.php . . . 9
7 Time.php . . . 10
8 api.php . . . 12
9 web.php . . . 12

Introducció

En aquest projecte final de l'assignatura hem creat una aplicació d'Internet on un usuari pot veure a temps real totes les dades dels seus servidors. Podrem veure percentatges d'espai lliure o utilitzat en els servidors, utilització de la xarxa, utilització de la CPU o també els processos que hi corren.

Per poder dur a terme aquest projecte, hem instal · lat els paquets llistats en el fitxer a les nostres màquines per: Instal · lar les llibreries necessàries per poder treballar amb la framework de Laravel i per poder obtenir mitjançant un script de Python3 les dades del servidor.

Arxiu que ens permet instal · lar tot el necessari per dur a terme el monitoratge i treballar amb la frame:

```
#! /bin/sh
sudo apt-get update
sudo apt-get upgrade
sudo apt-get install composer
sudo apt-get install -y git curl wget zip unzip
sudo apt-get install apache2
sudo a2enmod rewrite
sudo systemctl restart apache2
sudo apt-get install mysql-server
mysql_secure_installation
sudo add-apt-repository -y ppa:ondrej/php
sudo apt-get update
sudo apt-get install -y php7.2 php7.2-fpm libapache2-mod-php7.2 php7.2-cli php7.2-
    curl php7.2-mysql php7.2-sqlite3 php7.2-gd php7.2-xml php7.1-mcrypt php7.2-
    mbstring php7.2-iconv
curl -sS https://getcomposer.org/installer | sudo php -- --install-dir=/usr/local/bin
    --filename=composer
sudo chown -R $USER $HOME/.composer
sudo add-apt-repository ppa:jonathonf/python-3.6
sudo apt update
sudo apt install python3.6
sudo apt install python3.6-dev
sudo apt install python3.6-venv
wget https://bootstrap.pypa.io/get-pip.py
sudo python3.6 get-pip.py
sudo ln -s /usr/bin/python3.6 /usr/local/bin/python3
sudo ln -s /usr/local/bin/pip /usr/local/bin/pip3
sudo pip3.6 install psutil
sudo apt-get update
sudo apt-get upgrade
composer install
cp .env.example .env
php artisan key:generate
```

Codi 1: installer

Aquest arxiu instal · larà a grans trets: Python3.6 en l'última versió disponible i la llibreria psutils per poder obtenir les dades del monitoratge; PHP a una versió suficient per poder usar Laravel, MySQL per poder gestionar la base de dades, el composer també per Laravel i finalment Apache2 per si es vol configurar aquest pel projecte.

Python

Per obtenir les dades del sistema que monitorarem a temps real hem utilitzat la llibreria psutils. Hem implementat un script de Python3.6 que recull aquestes dades.

Per afegir un nou servidor al sistema de monitoratge, cal executar aquest arxiu amb els següents paràmetres:

UBUNTU

```
./monitoring API_TOKEN SERVER_NAME INTERVAL(sec)
```

WINDOWS

```
./monitoring.exe API_TOKEN SERVER_NAME INTERVAL(sec)
```

API_TOKEN: Obligatori. Serveix per autentificar-te al servidor.

SERVER_NAME: Obligatori. Serveix per determinar el nom del servidor a utilitzar. En crea un nou si no existeix.

INTERVAL(sec): Opcional. Serveix per indicar el temps d'actualització de les dades. Per defecte són 60sec.

Aquestes dades obtingudes per pas de paràmetre, juntament amb les dades obtingudes amb la llibreria psutils, s'envien amb post en JSON a una URL.

```python
import psutil, requests, json, time, sys, socket, platform

# python3.6
# psutil
# pip3

def dsk():
    prt = psutil.disk_partitions()
    df = {}
    for i in prt:
        di = { i.device : {
        'mount_point': i.mountpoint,
        'file_system': i.fstype,
        'total': psutil.disk_usage(i.mountpoint).total,
        'used': psutil.disk_usage(i.mountpoint).used,
        'free': psutil.disk_usage(i.mountpoint).free,
        'percent': psutil.disk_usage(i.mountpoint).percent
        } }
        df = {**df, **di}
    return df

def conStats():
    con = psutil.net_connections()
    stable = 0
    listen = 0
    for i in con:
        cond = i.status
        if cond == 'ESTABLISHED':
            stable += 1
        elif cond == 'LISTEN':
            listen += 1
    return { 'total': len(con), 'stable': stable, 'listen': listen }

def addrs():
    ad = psutil.net_if_addrs()
    fl = {}
    for e in ad.keys():
        for i in ad[e]:
            if i.family == socket.AF_INET:
                sl = { e : i.address }
                fl = {**fl, **sl}
```

```
                break
    return fl

def data():

    url = 'https://moni.erik.cat/api/data?api_token=' + sys.argv[1]
    payload = {
        'server': sys.argv[2],
        'os': platform.system(),
        'version': platform.release(),
        'platform': platform.platform(),
        'processor': platform.processor(),
        'node': platform.node(),
        'data': {
            'cpu': {
                'cores': psutil.cpu_count(),
                'percent': psutil.cpu_percent(), # Total
                'frequency': psutil.cpu_freq().current,
                'min_frequency': psutil.cpu_freq().min,
                'max_frequency': psutil.cpu_freq().max
            },
            'mem': { # System
                'total': psutil.virtual_memory().total,
                'available': psutil.virtual_memory().available,
                'used': psutil.virtual_memory().used,
                'percent': psutil.virtual_memory().percent,
                'free': psutil.virtual_memory().free,
            },
            'disks': dsk(),
            'net': conStats(),
            'addresses': addrs(),
            'pids': len(psutil.pids())
        }
    }

    try:
        r = requests.post(url, json = payload, headers = {'Accept': 'application/json
    '})
        if r.status_code == 200:
            print('Data sent to the server')
        else:
            print(r.json())
            print ('There was an error with the request.')
    except:
        print ('There was an error with the request.')

def main(interval):
    while True:
        d = data()
        time.sleep(interval)

if __name__ == "__main__":
    if len(sys.argv) == 1:
        print('The api_token is missing, please enter the API token as the first
    parameter')
        sys.exit(0)
    if len(sys.argv) == 2:
        print('The server_name is missing, please enter a server name as a second
    parameter. It will be created if it did not exist')
        sys.exit(0)
    if len(sys.argv) == 3:
        interval = 60
    elif len(sys.argv) > 3:
        interval = sys.argv[3]
        try:
            interval = int(interval)
        except:
            print('Wrong interval')
```

```
107             sys.exit(0)
108     print('Interval set to ' + str(interval) + ' seconds')
109 main(interval)
```

Codi 2: monitoring.py

PHP

Malgrat podríem fer aquest projecte únicament amb aquest llenguatge però, l'utilitzarem principalment per utilitzar la framework.

Principalment usarem la comanda: php artisan *comand* per poder fer moltes coses.

Laravel, PHP i MySQL

Un cop ens van arribant les dades des de l'arxiu de Python, les hem de classificar segons el que vulguem veure. Hem creat diferents mòduls on a cada mòdul farem diferents relacions i classificarem les dades:

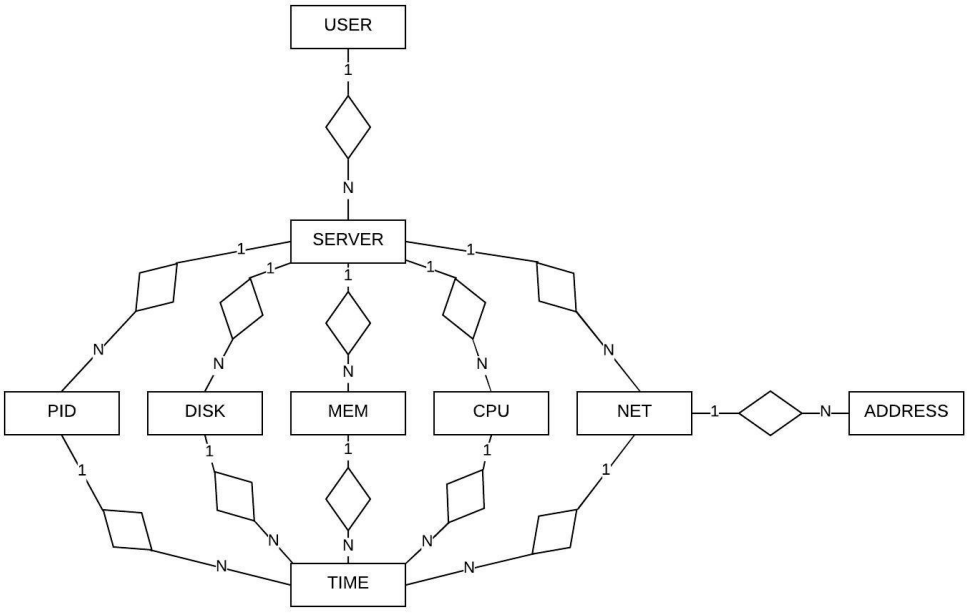

Figura 1: Esquema moduls

Gracies a l'ORM de laravel, podrem traduir els diferents mòduls a una base de dades amb MySQL i es podrà gestionar la base de dades.

User

Principalment tenim el mòdul **user**, on demanem a l'usuari que ens doni un nom, el seu email i la contrasenya, i crearem un token (anoment api_token).

Per fer-ho segur tenim la funció *setPasswordAttribute*, que aquí creem el hash de la contrasenya. Per últim veiem l'última funció *servers*, on fem la relació amb l'esquema anterior, és a dir, un usuari pot tenir diversos servidors.

```
1 <?php
2
3 namespace App;
4
5 use Illuminate\Notifications\Notifiable;
6 use Illuminate\Foundation\Auth\User as Authenticatable;
```

```php

class User extends Authenticatable
{
    use Notifiable;

    /**
     * The attributes that are mass assignable.
     *
     * @var array
     */
    protected $fillable = [
        'name', 'email', 'password', 'api_token'
    ];

    /**
     * The attributes that should be hidden for arrays.
     *
     * @var array
     */
    protected $hidden = [
        'password', 'remember_token',
    ];

    /**
     * Mutator to automatically hash the user password.
     *
     * @param string $value
     * @return void
     */
    public function setPasswordAttribute($value)
    {
        $this->attributes['password'] = bcrypt($value);
    }

    /**
     * Return the user servers.
     *
     * @return Collection
     */
    public function servers()
    {
        return $this->hasMany(Server::class);
    }
}
```

Codi 3: User.php

Server

En el modul **server** primerament rebrem el món, el sistema operatiu actiu, la seva versió, el processador, node i la plataforma utilitzada, i tot un seguit de funcions on podem veure la relació amb la resta de mòduls. Primer tenim la relació amb user, on veiem que el servidor només correspon a un únic usuari, però les següents relacions podem comprobar com per exemple un servidor pot tenir diversos processos (*return $this->hasMany(Pid::class);*), i així amb els mòduls **net**, **Disk**, **CPU**, **Mem** i **Net**.

```php
<?php

namespace App;

use Illuminate\Database\Eloquent\Model;

class Server extends Model
{
    /**
```

```
10          * Mass assignable attributes.
11          *
12          * @var array
13          */
14         public $fillable = [
15             'name', 'os', 'version', 'processor', 'node', 'platform'
16         ];
17
18         /**
19          * Returns the server user.
20          *
21          * @return App\User
22          */
23         public function user()
24         {
25             return $this->belongsTo(User::class);
26         }
27
28         /**
29          * Returns the server PIDs.
30          *
31          * @return App\Pid
32          */
33         public function pids()
34         {
35             return $this->hasMany(Pid::class);
36         }
37
38         /**
39          * Returns the server NET.
40          *
41          * @return App\Net
42          */
43         public function nets()
44         {
45             return $this->hasMany(Net::class);
46         }
47
48         /**
49          * Returns the server memory.
50          *
51          * @return App\Mem
52          */
53         public function mems()
54         {
55             return $this->hasMany(Mem::class);
56         }
57
58         /**
59          * Returns the server CPU.
60          *
61          * @return App\Cpu
62          */
63         public function cpus()
64         {
65             return $this->hasMany(Cpu::class);
66         }
67
68         /**
69          * Returns the server DISK.
70          *
71          * @return App\Disk
72          */
73         public function disks()
74         {
75             return $this->hasMany(Disk::class);
76         }
77 }
```

Codi 4: Server.php

Net, Pid, Disk, CPU, Mem, Net

En aquests mòduls pràcticament farem el mateix en tots. De tota la informació que ens arriba agafem la que ens sigui necessària segons el modul, i fem la serva relació amb la resta de mòduls. Per exemple en el mòdul **Disk** ens interès el seu espai lliure, total, ocupat i el percentatge corresponent, i després fem la relació amb el mòdul **Server** i **Time**

```php
<?php

namespace App;

use Illuminate\Database\Eloquent\Model;

class Disk extends Model
{
    /**
     * Mass assignable attributes.
     *
     * @var array
     */
    public $fillable = [
        'device', 'mount_point', 'file_system', 'total', 'used', 'free', 'percent'
    ];

    /**
     * Returns the disk server.
     *
     * @return App\Server
     */
    public function server()
    {
        return $this->belongsTo(Server::class);
    }

    /**
     * Return the disk time.
     *
     * @return App\Time
     */
    public function time()
    {
        return $this->belongsTo(Time::class);
    }
}
```

Codi 5: Disk.php

Address

Mostra totes les adreces IP del teu servidor amb les interfícies corresponent. Aquest és relaciona amb el mòdul **Net**, ja que una xarxa pot tenir diferents IP.

```php
<?php

namespace App;

use Illuminate\Database\Eloquent\Model;

```

```php
class Address extends Model
{
    /**
     * Mass assignable attributes.
     *
     *@var array
     */
    public $fillable = [
        'name', 'ip'
    ];

    /**
     * Returns the user.
     *
     * @return void
     */
    public function user()
    {
        return $this->belongsTo(Net::class);
    }
}
```

Codi 6: Address.php

Controller

```php
<?php

namespace App\Http\Controllers;

use App\User;
use Illuminate\Http\Request;

class MoniController extends Controller
{
    /**
     * Stores the server information.
     *
     * @param Request $request
     * @return array
     */
    public function data(Request $request)
    {
        $user = auth('api')->user();

        $server = $user->servers()->where('name', $request->get('server'))->
    firstOrCreate([
            'name'      => $request->get('server'),
            'os'        => $request->get('os'),
            'version'   => $request->get('version'),
            'node'      => $request->get('node'),
            'processor' => $request->get('processor'),
            'platform'   => $request->get('platform'),
        ]);

        $server->update([
            'os'        => $request->get('os'),
            'version'   => $request->get('version'),
            'node'      => $request->get('node'),
            'processor' => $request->get('processor'),
            'platform'   => $request->get('platform'),
        ]);

        $data = $request->get('data');

```

```php
        // CPU
        $server->cpus()->updateOrcreate([
            'time_id' => null
        ], [
            'cores'         => $data['cpu']['cores'],
            'percent'       => $data['cpu']['percent'],
            'frequency'     => $data['cpu']['frequency'],
            'min_frequency' => $data['cpu']['min_frequency'],
            'max_frequency' => $data['cpu']['max_frequency']
        ]);

        // Disks
        $server->disks()->where('time_id', null)->delete();

        foreach ($data['disks'] as $key => $disk) {
            $server->disks()->create([
                'device'       => $key,
                'mount_point'   => $disk['mount_point'],
                'file_system'   => $disk['file_system'],
                'total'         => round($disk['total'] * 9.31e-10),
                'used'          => round($disk['used'] * 9.31e-10),
                'free'          => round($disk['free'] * 9.31e-10),
                'percent'       => $disk['percent']
            ]);
        }

        // Mem
        $server->mems()->updateOrcreate([
            'time_id' => null
        ], [
            'total'     => round($data['mem']['total'] * 9.31e-10),
            'available' => round($data['mem']['available'] * 9.31e-10),
            'used'      => round($data['mem']['used'] * 9.31e-10),
            'percent'   => $data['mem']['percent'],
            'free'      => round($data['mem']['free'] * 9.31e-10)
        ]);

        // Net
        $net = $server->nets()->updateOrcreate([
            'time_id' => null
        ], [
            'total' => $data['net']['total'],
            'stable' => $data['net']['stable'],
            'listen' => $data['net']['listen']
        ]);

        $net->addresses()->delete();
        foreach ($data['addresses'] as $key => $address) {
            $net->addresses()->create([
                'name'  => $key,
                'ip'    => $address
            ]);
        }

        // Pid
        $server->pids()->updateOrcreate([
            'time_id' => null
        ],[
            'number' => $data['pids']
        ]);

        return [
            'response' => 'ok'
        ];
    }
}
```

Codi 7: Time.php

Routes:

En el projecte de Laravel usar 2 routes una per obtenir la interfície i un altre pel backend.

Route de backend:

```php
<?php

use Illuminate\Http\Request;

/*
|--------------------------------------------------------------------------
| API Routes
|--------------------------------------------------------------------------
|
| Here is where you can register API routes for your application. These
| routes are loaded by the RouteServiceProvider within a group which
| is assigned the "api" middleware group. Enjoy building your API!
|
*/

Route::post('/login', 'AppController@login');
Route::post('/register', 'AppController@register');

Route::middleware('auth:api')->group(function () {
    Route::post('/servers', 'AppController@servers');
    Route::post('/servers/{server}', 'AppController@server');
    Route::post('/data', 'MoniController@data');
});
```

Codi 8: api.php

Les línies de Route::post, envien l'execució amb un metode post a MoniController@lacommanda

El Route::middleware('auth:api') protegeix l'execucció que sigui l'usuari correcte mitjançant el token.

Route de la interfície:

```php
<?php

/*
|--------------------------------------------------------------------------
| Web Routes
|--------------------------------------------------------------------------
|
| Here is where you can register web routes for your application. These
| routes are loaded by the RouteServiceProvider within a group which
| contains the "web" middleware group. Now create something great!
|
*/

Route::get('/{any}', 'AppController@index')->where('any', '.*');
```

Codi 9: web.php

Mitjançant la comanda post el que fa és obtenir la interfície.

Web socket

Mitjançant el servei pusher, avisa que hi ha les dades noves mitjançant web sockets.

Interfície

La interfície està basada en VueJS, creant un virtualDOM que és fàcilment actualitzable des de javascript. Com a disseny es fa servir vuetify per crear una interfície interactiva amb poca feina. Es fa servir websockets per escoltar i reaccionar als events.

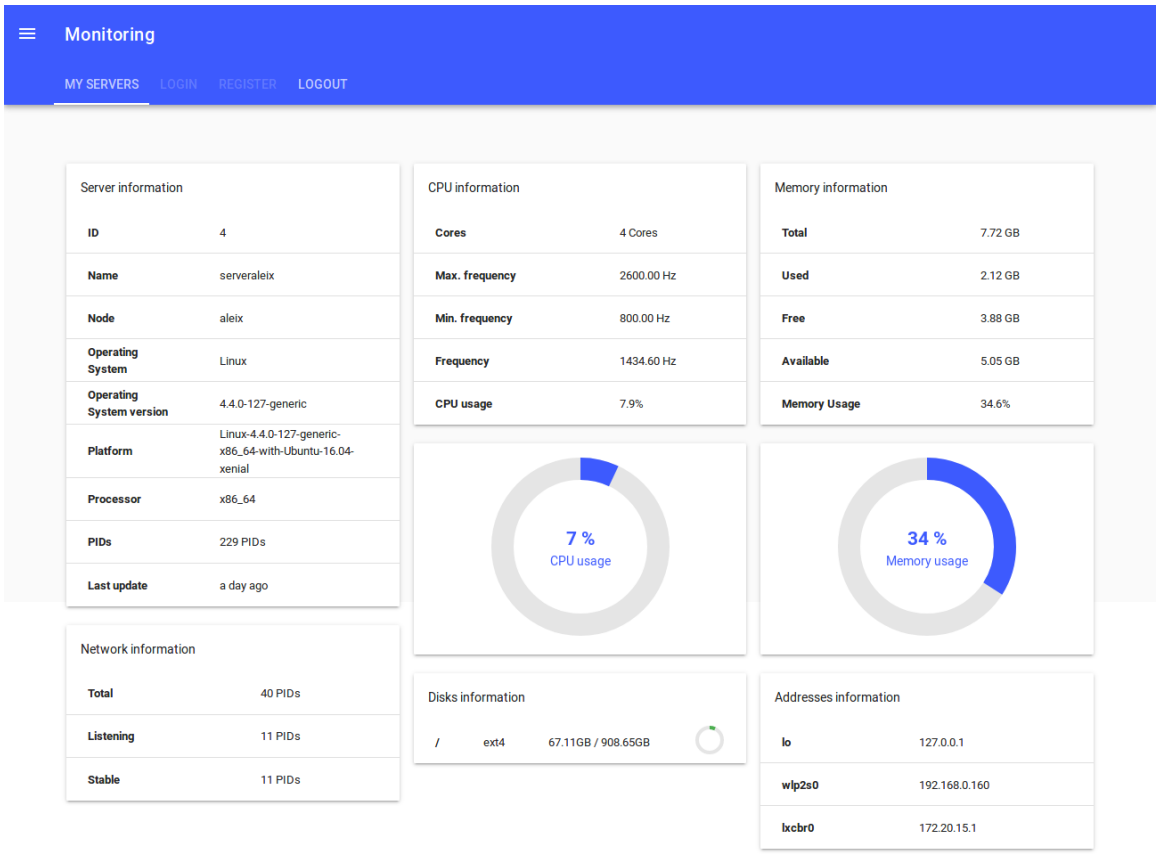

Figura 2: Interfície web

Autentificació

Es fa servir autentificació a través de tokens de 60 caràcters de longitud, generats per usuari al moment del registre.