



UNIVERSIDAD DE HUELVA

ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA
CIENCIAS DE LA COMPUTACIÓN E INTELIGENCIA ARTIFICIAL

TÍTULO DEL TRABAJO:

**Implementación de un geolocalizador de IPs haciendo
uso de las tecnologías proporcionadas por Amazon
Web Services (AWS).**

Trabajo subministrado por Juan Carlos de la Prada de Haro y Sergio Rodríguez García para la asignatura Cloud Computing del Máster Oficial en Ingeniería Informática.

Profesor:
José Carpio Cañada

Contenidos

Apartados	Página
Introducción	I
1. Investigación de las distintas posibilidades que ofrece AWS para implementar servicios de alta disponibilidad.	1
2. Elección de los servicios AWS e implementación del geolocalizador de IPs	3
2.1. Elección de los servicios AWS a utilizar	3
2.1.1. Creación del servicio Amazon RDS	3
2.2. Elección del servidor web para el alojamiento del geolocalizador	8
2.3. Elección del sistema de gestión de la base de datos	9
2.4. Elección del framework para el desarrollo de la API	9
2.5. Implementación del geolocalizador de IPs	11
3. Sistema AWS en producción y posibles problemáticas del mismo	14
4. Conclusión	17
Referencias	II

Introducción

A lo largo de este documento se detallará como llevar a cabo la implementación de un sistema de geolocalización de IPs capaz de abastecer todas las peticiones realizadas por los usuarios.

Para afrontar el problema de la demanda que se realice de nuestro servicio, se piensa crear un servicio auto escalable de alta disponibilidad, ajustando los distintos parámetros del balanceador de carga. Para ello se va a aplicar el conocimiento aprendido en las sesiones anteriores sobre manejo de instancias del servicio EC2 de AWS con sistema operativo Linux. Además se investigarán las diferentes posibilidades que ofrece AWS para implementar servicios de alta disponibilidad incluyendo las distintas opciones estudiadas en otras sesiones. En el caso de utilizar instancias, estas serán de tipo t2.micro.

El servicio se implementará utilizando una API REST cuya base de datos es proporcionada por IP2Location.

Con el fin de realizar un correcto desarrollo de la documentación se ha optado por utilizar el sistema de composición de textos \LaTeX

IP2LOCATION™



1. Investigación de las distintas posibilidades que ofrece AWS para implementar servicios de alta disponibilidad.

Los distintos servicios que se han tenido en cuenta antes de seleccionar los elementos de AWS que van a comprender nuestro sistema de geolocalización son:

- **Amazon Elastic Compute Cloud (Amazon EC2):** servicio web que proporciona capacidad de cómputo con tamaño modificable en la nube, configurable a partir de una interfaz sencilla. Además Amazon EC2 reduce el tiempo necesario para lanzar nuevas instancias de servidor en cuestión de minutos, lo que permite escalar rápidamente la capacidad, ya sea aumentándola o reduciéndola.



Figura 1.1: Logo Amazon EC2.

- **Auto Scaling:** permite gestionar la disponibilidad de la aplicación habiendo previamente definido ciertas condiciones, aumentando o reduciendo así la capacidad de Amazon EC2. Esto cubre el aseguramiento de que se están ejecutando la cantidad de instancias deseadas, el incremento automático del número de instancias en los periodos de mayor demanda y la reducción automática del número de instancias en los periodos de declive de la demanda.

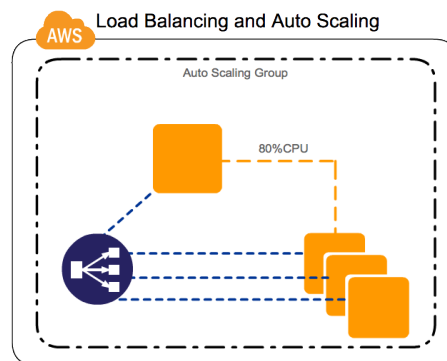


Figura 1.2: Ejemplo Auto Scaling.

- **Amazon ElastiCache:** servicio web que facilita la implementación, el funcionamiento y el escalado de una caché o almacén de datos en memoria en la nube. El servicio mejora el desempeño de las aplicaciones web, lo que le permite recuperar información rápidamente de almacenes de datos en memoria. Además proporciona un sistema resistente que reduce el riesgo de sobrecargas en las bases de datos, que ralentizan los tiempos de carga del sitio web y la aplicación.



Figura 1.3: Logo Amazon ElastiCache.

- **Amazon Relational Database Service (Amazon RDS):** sencillo configurar, utilizar y escalar una base de datos relacional en la nube. Proporciona capacidad rentable y de tamaño modificable y además administra dificultosas tareas de administración de la BD, permitiendo que nos centremos en el desarrollo de nuestras aplicaciones. Amazon RDS permite elegir de entre seis motores de bases de datos; Amazon Aurora, PostgreSQL, MySQL, MariaDB, Oracle y Microsoft SQL Server.



Figura 1.4: Logo Amazon RDS.

- **Amazon DynamoDB:** servicio de base de datos NoSQL rápido y flexible para todas las aplicaciones que necesiten latencias constantes y de apenas unos milisegundos a cualquier escala. Es una base de datos totalmente administrada en la nube, compatible con modelos de almacenamiento de datos de valor de clave y de documentos.



Figura 1.5: Logo Amazon DynamoDB.

2. Elección de los servicios AWS e implementación del geolocalizador de IPs

2.1. Elección de los servicios AWS a utilizar

Para proseguir con la realización de nuestro sistema al investigar los distintos elementos que nos proporciona AWS nos hemos podido percatar de que era necesario seleccionar al menos un servicio de computación y como complemento adicional seleccionar o no un servicio de base de datos. Por tanto hemos partido de la idea inicial de añadir un servicio de base de datos, ya que al utilizar únicamente elementos de computo no se pueden realizar modificaciones en la BD sin asegurar su consistencia en accesos concurrentes, intentando una aproximación lo más cercana posible al mundo real donde los datos se crean, modifican y destruyen.

Esta razón nos ha requerido meditar que servicio de BD escoger, si RDS o DynamoDB. Leyendo las preguntas frecuentes de DynamoDB encontramos la pregunta “¿Cuándo debo utilizar Amazon DynamoDB en lugar de un motor de base de datos relacional en Amazon RDS o Amazon EC2?” donde podemos resaltar alguna de sus líneas donde se menciona que no ofrecen toda la funcionalidad de una base de datos relacional, no admite consultas relacionales (como, por ejemplo, uniones) y por tanto tampoco transacciones complejas. Es por esto que a pesar de la facilidad de uso que nos proporciona DynamoDB hemos decidido decantarnos por Amazon RDS para no capar de funcionalidades nuestro sistema.

Finalmente nos hemos decantado por el servicio de computación EC2 ya que en su página principal mencionan que es un servicio diseñado para utilizarse con otros servicios de Amazon Web Services como Amazon RDS para proporcionar una solución completa de informática, procesamiento de consultas y almacenamiento para una gran variedad de soluciones, no encontrando dicha información en la página principal de ElastiCache. Queda destacar que como mecanismo para afrontar la oleada de peticiones de los usuarios se utilizará como servicio el de Auto Scaling.

Como en las anteriores sesiones de prácticas se han utilizado tanto el servicio de Amazon EC2 como el de Auto Scaling pero no el de Amazon RDS, en la siguiente subsección se entrará en detalle de como se crea una instancia de dicho servicio.

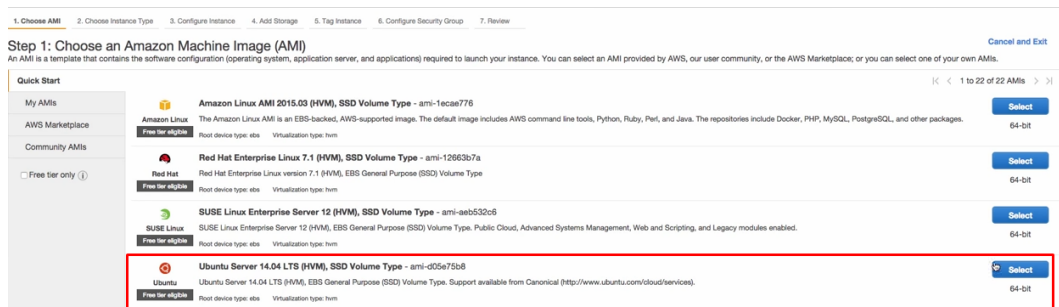
2.1.1. Creación del servicio Amazon RDS

En esta subsección se detallarán los puntos a seguir para llegar a lanzar una instancia de Amazon RDS accesible mediante una instancia Amazon EC2 haciendo uso de phpMyAdmin. Para ello nos hemos basado en el video tutorial “Connecting To RDS MySQL Using

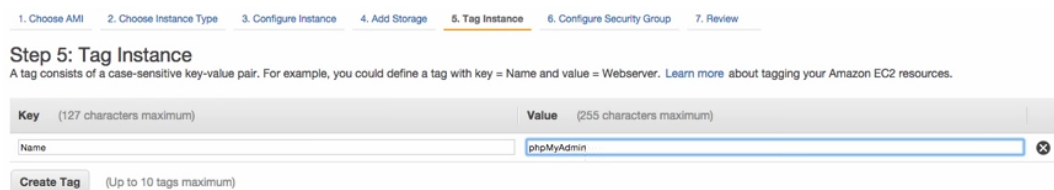
phpMyAdmin (EC2, RDS)” del propio canal de youtube de Amazon Web Services “AWS Tutorial Series”.

Los pasos a seguir son:

1. Creamos una instancia EC2 cuyo sistema operativo será Ubuntu y a la cuál denominaremos phpMyAdmin como podemos observar en las siguiente figura.



(a) Elección del sistema operativo.



(b) Definiendo el nombre de la instancia.

Figura 2.1: Nueva instancia Amazon EC2 con SO Ubuntu.

2. Salimos del menú EC2 Dashboard y accedemos al menú RDS Dashboard. A continuación se puede observar la forma del mismo.

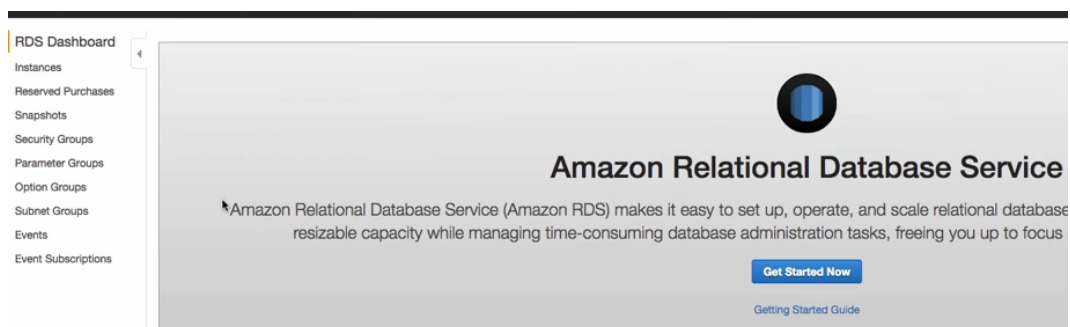


Figura 2.2: RDS Dashboard.

3. Creamos un nuevo DB Subnet Group con los datos que se muestran en la siguiente figura.

Create DB Subnet Group

To create a new Subnet Group give it a name, description, and select an existing VPC below. Once you select a VPC you will be able to add subnets related to that VPC.

Name ⓘ

Description ⓘ

VPC ID ⓘ

Add Subnet(s) to this Subnet Group. You may add subnets one at a time below or **add all the subnets** related to this VPC. You may make additions/edits after this group is created. A minimum of 2 subnets is required.

Availability Zone ⓘ

Subnet ID ⓘ

Availability Zone	Subnet ID	CIDR Block	Action
us-east-1c	subnet-71836628	172.30.2.0/24	<input type="button" value="Remove"/>
us-east-1a	subnet-b9151091	172.30.0.0/24	<input type="button" value="Remove"/>
us-east-1b	subnet-2c45bd5b	172.30.1.0/24	<input type="button" value="Remove"/>

Figura 2.3: Creación de un DB Subnet Group.

4. Creamos una instancia de Amazon RDS, para ello elijeremos como motor de BD el de MySQL.

Select Engine

To get started, choose a DB Engine below and click Select.


MySQL


PostgreSQL


ORACLE

MySQL
MySQL Community Edition

Figura 2.4: Selección del motor de bases de datos MySQL.

Posteriormente introduciremos la configuración de la misma tal cuál podemos observar en la figura 2.5.

Specify DB Details

Instance Specifications

DB Engine: **mysql**

License Model: **general-public-license**

DB Engine Version: **5.6.22**

Review the **Known Issues/Limitations** to learn about potential compatibility issues with specific database versions.

DB Instance Class: **db.t1.micro — 1 vCPU, 0.613 GiB RAM**

Multi-AZ Deployment: **No**

Storage Type: **General Purpose (SSD)**

Allocated Storage*: **5** GB

Provisioning less than 100 GB of General Purpose (SSD) storage for high throughput workloads could result in higher latencies upon exhaustion of the initial General Purpose (SSD) IO credit balance. [Click here](#) for more details.

Settings

DB Instance Identifier*: **phpMyAdmin**

Master Username*: **phpMyAdmin**

Master Password*: *********

Confirm Password*: *********

Configure Advanced Settings

Network & Security

This instance will be created with the new Certificate Authority rds-ca-2015. If you are using SSL to connect to this instance, you should use the [new certificate bundle](#). [Learn more here](#)

VPC: **vpc-89af74ec**

Subnet Group: **phpmyadmin**

Publicly Accessible: **No**

Availability Zone: **No Preference**

VPC Security Group(s): **Create new Security Group**
EC2 - Wordpress Production (VPC)
RDS - Wordpress (VPC)
Wide Open (VPC)

Database Options

Database Name: **awstutorialseries**

Note: If no database name is specified then no initial MySQL database will be created on the DB Instance.

Database Port: **3306**

DB Parameter Group: **default:mysql5.6**

Option Group: **default:mysql-5-6**

Enable Encryption: **No**

The selected Engine or DB Instance Class does not support storage encryption.

(a) Especificando los detalles de la BD.

(b) Especificando la configuración avanzada

Figura 2.5: Detalles de la instancia.

Finalmente ya tendremos lanzada nuestra instancia Amazon RDS.

Launch DB Instance

Show Monitoring

Instance Actions

Filter: All Instances

Search DB Instances...

	DB Instance	VPC	Multi-AZ	Class	Status	Maintenance	Storage Type	Storage	Security Groups	Engine
	phpmyadmin	vpc-89af74ec	No	db.t1.micro	available	None	General Purpose (SSD)	5 GB	Wide Open (active)	mysql

Figura 2.6: Instancia Amazon RDS.

5. Entramos en nuestra instancia EC2 para configurarla con los siguientes comandos:

```
sudo su
apt-get update
apt-get upgrade -y
apt-get dist-upgrade -y
apt-get autoremove -y
apt-get install apache2 php5 php5-cli php5-fpm php5-gd libssh2-
    php libapache2-mod-php5 php5-mcrypt mysql-server php5-mysql
    git unzip zip postfix php5-curl mailutils php5-json
    phpmyadmin -y
php5enmod mcrypt

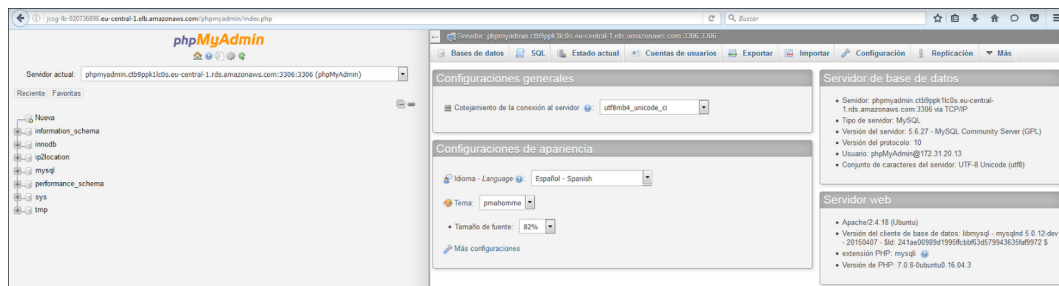
nano /etc/apache2/sites-enabled/000-default.conf
--ADD LINE--
Include /etc/phpmyadmin/apache.conf

service apache2 restart
```

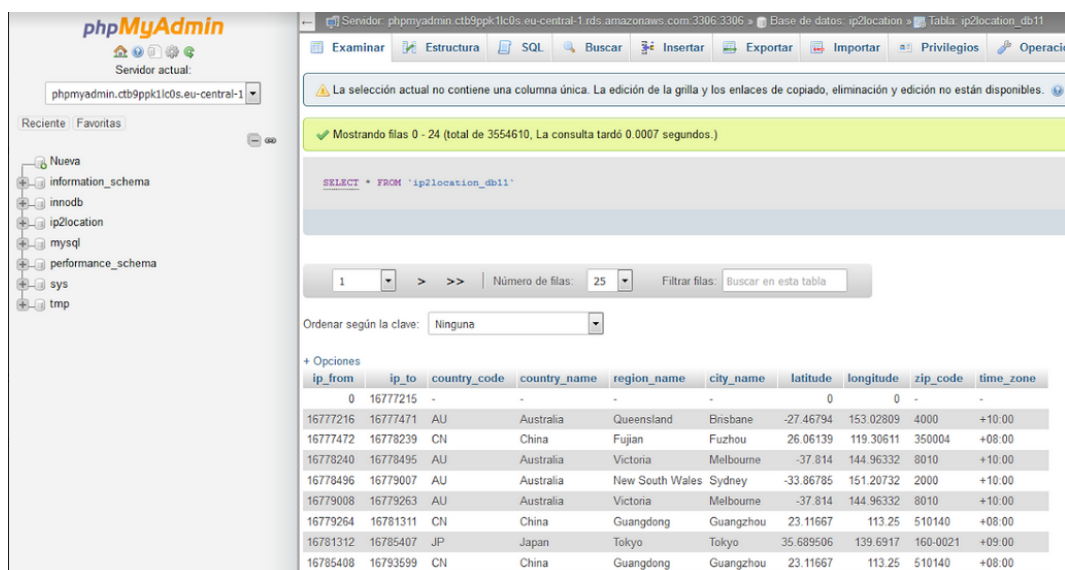
Posteriormente añadiremos las siguientes líneas al fichero “config.ini.php”:

```
nano /etc/phpmyadmin/config.ini.php
--ADD LINES BELOW THE PMA CONFIG AREA--
$i++;
$cfg[ 'Servers' ][ $i ][ 'host' ]          = 'phpmyadmin.ctb9ppk1lc0s.
    eu-central-1.rds.amazonaws.com';
$cfg[ 'Servers' ][ $i ][ 'port' ]          = '3306';
$cfg[ 'Servers' ][ $i ][ 'socket' ]        = '';
$cfg[ 'Servers' ][ $i ][ 'connect_type' ]  = 'tcp';
$cfg[ 'Servers' ][ $i ][ 'extension' ]     = 'mysql';
$cfg[ 'Servers' ][ $i ][ 'compress' ]      = FALSE;
$cfg[ 'Servers' ][ $i ][ 'auth_type' ]     = 'config';
$cfg[ 'Servers' ][ $i ][ 'user' ]          = 'phpMyAdmin';
$cfg[ 'Servers' ][ $i ][ 'password' ]      = 'phpMyAdmin';
```

Con esto finaliza la puesta en marcha de nuestro sistema, pudiendo así acceder a nuestra BD de la instancia RDS mediante la máquina EC2 haciendo uso de phpMyAdmin.



(a) Home de phpMyAdmin usando como servidor la instancia Amazon RDS.



(b) Base de datos de la geolocalización de IPs cargadas en la instancia Amazon RDS

Figura 2.7: Amazon RDS a través de phpMyAdmin.

2.2. Elección del servidor web para el alojamiento del geolocalizador

Nuestra elección ha sido la de utilizar el servidor web Apache, ya que es mundialmente conocido por todos.

Cabe destacar que una vez realizado el geolocalizador y debido a algunos problemas que mencionaremos en el siguiente apartado nos detuvimos un poco más y miramos otros servidores, encontrando así el servidor web HTTP Nginx, que está pensado para utilizarse como proxy inverso, el cuál trabajando de este modo se utiliza para equilibrar la carga entre servidores back-end y como caché en servidores back-end lentos. Las características por las que destaca Nginx son las siguientes:

- Es ligero: Reduce el consumo de RAM.

- Es multiplataforma.
- Puede ser usado como caché para mejorar la eficiencia de las aplicaciones
- Trabaja como balanceador de carga.

Nos pareció mejor que Apache e intentamos instalar Ruby on Rails en él, pero al tener una instancia con el servidor Apache ya configurado al completo y no prescindir de todo el tiempo que nos gustaría para lanzar máquinas nos decidimos finalmente por seguir con Apache.

2.3. Elección del sistema de gestión de la base de datos

Al haber elegido como servicio de almacenamiento el de Amazon RDS, tenemos que seleccionar alguno de los siguientes SGBD; MySQL, Oracle, PostgreSQL, MariaDB, Microsoft SQL Server o Amazon Aurora.

Nuestra elección ha sido la de instalar el SGBD MySQL ya que esta considerada como el SGBD open source más popular del mundo y una de las más populares, además de que como vimos en el tutorial de instalación de instancias para el servicio de Amazon RDS permite administrarlo mediante phpMyAdmin.



Figura 2.8: Logo MySQL.

2.4. Elección del framework para el desarrollo de la API

Para la elección del framework estudiamos y probamos varios candidatos. Los principales fueron para el lenguaje de programación PHP, estos son; Slim y Laravel.

Slim es un micro framework para PHP que nos permite realizar rápidamente aplicaciones web y APIs. Sus características son las siguientes:

- Creador de rutas muy potente.
 - Soporta métodos HTTP estándar y personalizados.
 - Parámetros de ruta con comodines y condiciones.
 - Redirecciones de rutas, paros y saltos.
- Renderizado de plantillas y vistas personalizadas.
- Mensajes flash.

- Caché HTTP.
- Logging de accesos personalizado.
- Configuración sencilla.

Laravel es un framework para PHP de código abierto que nos facilita el desarrollo de aplicaciones y servicios web. Las ventajas que nos ofrece son:

- Sistema de ruteo y RESTFul.
- Blade: Motor de plantillas.
- Peticiones Fluent.
- Curva de aprendizaje baja.
- Eloquent ORM.
- Soporte para caché.

Finalmente escogimos Slim debido tanto a su rapidez para el desarrollo de aplicaciones como a su tamaño. Si es cierto que incluso una vez realizado el código de la aplicación intentamos probar también con Laravel ya que no lograbamos cambiar la raíz del proyecto por ejemplo de `‘/index.php/ruta’` a `‘/ruta’` redirigiendo realmente todo a través del `index.php`, para lo cual también intentamos modificar algunos ficheros de Apache o modificar el fichero `‘.htaccess’` pero nada de esto resultó en éxito. Fue tal la frustración que se intentó incluso cambiar de lenguaje de programación a Ruby e instalar el framework Ruby on Rails. Pero finalmente y debido al tiempo que teníamos decidimos quedarnos con Slim ya que Ruby on Rails lanzaba un servidor local a la aplicación.

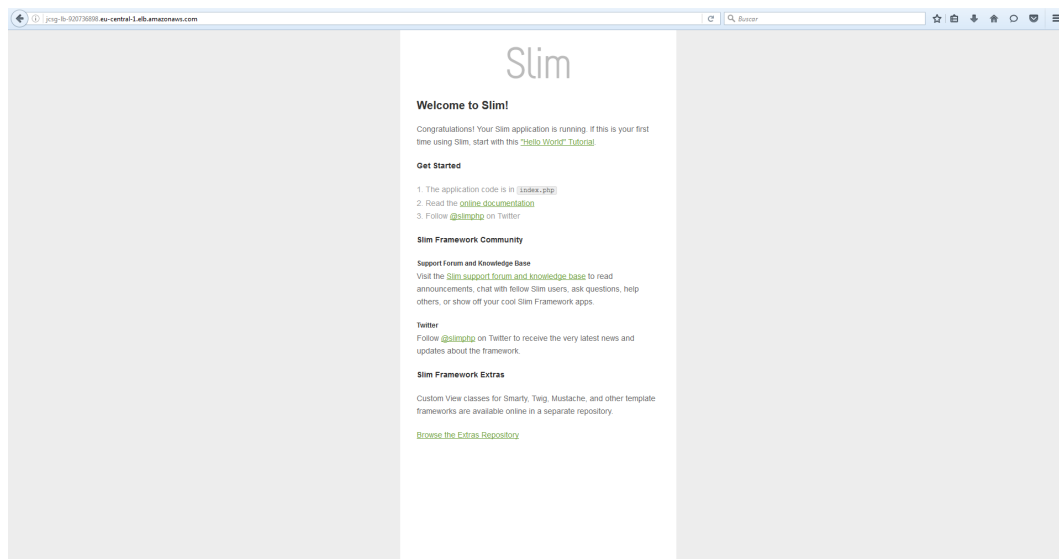


Figura 2.9: Framework Slim instalado.

2.5. Implementación del geolocalizador de IPs

La implementación del geolocalizador de IPs tiene como objetivo realizar una API Rest con acceso GET en los formatos; Json, CSV y XML.

Gracias al framework Slim podemos desarrollar la API con facilidad. A continuación se desglosan las funciones implementadas:

- Definiciones de la BD y función genérica para la obtención de IPs.

```
define( 'BD_SERVIDOR', $bd_servidor );
define( 'BD_USUARIO', 'phpMyAdmin' );
define( 'BD_PASSWORD', 'phpMyAdmin' );
define( 'BD_NOMBRE', 'ip2location' );

$conn = mysqli_connect( BD_SERVIDOR, BD_USUARIO, BD_PASSWORD,
    BD_NOMBRE );

function obtenerIP($ip) {
    if ( $ip == "random" ) {
        return rand(0, 255). "." . rand(0, 255). "." .
            rand(0, 255). "." . rand(0, 255);
    }
    return gethostbyname($ip);
}
```

- Función para obtener la salida de los datos en formato Json.

```
$app->get( '/json/:ip', function($ip) use($conn){
    $ip = obtenerIP($ip);
    $ip_sql = str_replace( '.', '', $ip);
    $sql = "SELECT country_code, country_name, region_name,
        city_name, zip_code, time_zone, latitude, longitude
        FROM ip2location_db11 WHERE " . $ip_sql . " BETWEEN
        ip_from_and_ip_to";
    $result = mysqli_query($conn, $sql);
    $datos;
    if (mysqli_num_rows($result) > 0) {
        $ip_row = array("ip" => $ip);
        $row = mysqli_fetch_assoc($result);
        $datos = array_merge($ip_row, $row);
        print json_encode($datos);
    }
    mysqli_close($conn);
});
```

- Función para obtener la salida de los datos en formato CSV.

```
$app->get( '/csv/:ip', function($ip) use($conn){
    $ip = obtenerIP($ip);
    $ip_sql = str_replace( '.', '', $ip);
    $sql = "SELECT country_code, country_name, region_name,
        city_name, zip_code, time_zone, latitude, longitude
        FROM ip2location_db11 WHERE " . $ip_sql . " BETWEEN
        ip_from_and_ip_to";
```

```

$result = mysqli_query($conn, $sql);
$datos;
if (mysqli_num_rows($result) > 0) {
    $ip_row = array("ip" => $ip);
    $row = mysqli_fetch_assoc($result);
    $datos = array_merge($ip_row, $row);
    $csv = "";
    $i = 0;
    $num_elem = count($datos);
    foreach ($datos as $clave => $valor){
        if ($i != $num_elem - 1) {
            $csv = $csv . $valor . ",";
        }
        else {
            $csv = $csv . $valor;
        }
        $i++;
    }
    echo $csv;
}
mysqli_close($conn);
});

```

- Función para obtener la salida de los datos en formato XML.

```

$app->get('/xml/:ip', function($ip) use($conn){
    $ip = obtenerIP($ip);
    $ip_sql = str_replace(' ', '', $ip);
    $sql = "SELECT country_code, country_name, region_name, city_name,
        zip_code, time_zone, latitude, longitude FROM
        ip2location_db11 WHERE " . $ip_sql . " BETWEEN ip_from and
        ip_to";
    $result = mysqli_query($conn, $sql);
    $datos;
    if (mysqli_num_rows($result) > 0) {
        $ip_row = array("ip" => $ip);
        $row = mysqli_fetch_assoc($result);
        $datos = array_merge($ip_row, $row);
        $xml = "<Response>";
        $num_elem = count($datos);

        foreach ($datos as $clave => $valor){
            $etiqueta = "";
            if ($clave == "ip") {
                $etiqueta = "IP";
            }
            else if ($clave == "country_code") {
                $etiqueta = "CountryCode";
            }
            else if ($clave == "country_name") {
                $etiqueta = "CountryName";
            }
            else if ($clave == "region_name") {
                $etiqueta = "RegionName";
            }
        }
    }
}

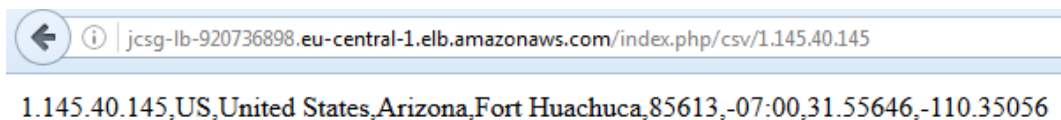
```

```

        else if ($clave == "city_name") {
            $etiqueta = "CityName";
        }
        else if ($clave == "zip_code") {
            $etiqueta = "ZipCode";
        }
        else if ($clave == "time_zone") {
            $etiqueta = "TimeZone";
        }
        else if ($clave == "latitude") {
            $etiqueta = "Latitude";
        }
        else if ($clave == "longitude") {
            $etiqueta = "Longitude";
        }
        echo $etiqueta;
        $xml = $xml . "<" . $etiqueta . ">" . $valor . "
            </" . $etiqueta . ">";
    }
    $xml = $xml . "</Response>";
    echo $xml;
}
mysqli_close($conn);
});

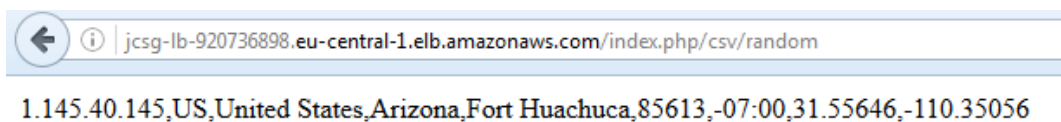
```

Por último vamos a mostrar en la siguiente figura algunas salidas producidas por dicho código para verificar que realmente funciona nuestra API.



1.145.40.145,US,United States,Arizona,Fort Huachuca,85613,-07:00,31.55646,-110.35056

(a) Formato CSV con IP



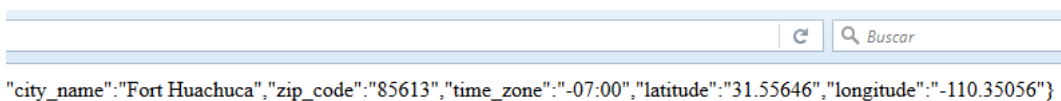
1.145.40.145,US,United States,Arizona,Fort Huachuca,85613,-07:00,31.55646,-110.35056

(b) Formato CSV con IP Random



{"ip":"236.110.64.68","country_code":"US","country_name":"United States","region_name":"Arizona","city_name":.

(c) Formato Json con IP Random (Parte 1)



"city_name":"Fort Huachuca","zip_code":"85613","time_zone":"-07:00","latitude":"31.55646","longitude":"-110.35056"}

(d) Formato Json con IP Random (Parte 2)

Figura 2.10: Salidas producidas en algunos de los formatos.

3. Sistema AWS en producción y posibles problemáticas del mismo

Lo que hicimos inicialmente fue crear un sistema de auto escalado con instancias EC2 cuyo código se conectaba a la BD de Amazon RDS pero vimos que antes de poder darle la oportunidad al auto escalado de incorporar nuevas instancias el sistema no respondía a partir de un cierto número de peticiones. Fue entonces cuando nos dimos cuenta de que no dependíamos tanto de las instancias de EC2 ni del auto escalado como de la instancia Amazon RDS cuya utilización de CPU ascendía como la espuma llegando a inhabilitar la instancia como podemos observar en la siguiente figura.

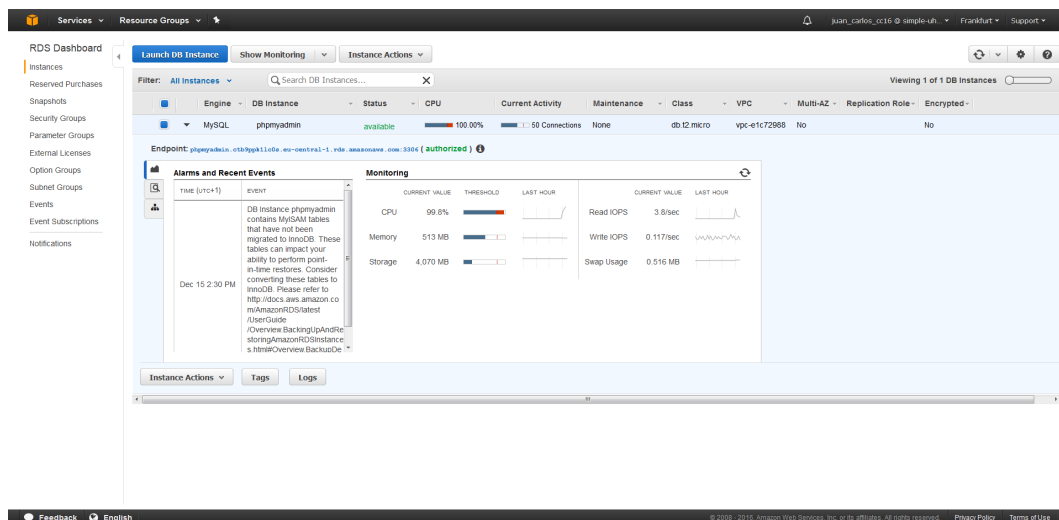


Figura 3.1: Amazon RDS con una instancia.

Por esto seguimos recabando información acerca de como escalar en el servicio de Amazon RDS, fue ahí donde leímos que para montar instancias de este servicio lo más recomendable era utilizar máquinas más grandes, pero aun así también mencionaba la posibilidad de crear una estructura maestro esclavo, con tu instancia inicial como maestro utilizándolo tanto para operaciones de lectura como de escritura y además crear esclavos que son simplemente instancias de sólo lectura.

A continuación se pueden ver el maestro con uno y dos esclavos.

The screenshot shows the AWS RDS console interface. The left sidebar contains navigation links like 'Instances', 'Reserved Purchases', 'Snapshots', etc. The main area displays the 'Launch DB Instance' page for a MySQL instance named 'phpmyadmin'. It shows the instance is 'available' with 1.50% CPU usage and 1 connection. Below this, the 'Replication Details' and 'Deployment DB Instances' are shown. The 'Deployment DB Instances' table lists the master instance 'phpmyadmin' and its replica 'phpmyadmin-replica'.

DB Instance	ROLE	ZONE	REPLICATION SOURCE	REPLICA LAG
phpmyadmin	master	eu-central-1b	-	-
phpmyadmin-replica	replica	eu-central-1b	phpmyadmin	0 ms

(a) Un maestro y un esclavo

	Engine	DB Instance	Status	CPU	Current Activity	Maintenance	Class	VPC	Multi-AZ	Replication Role
<input checked="" type="checkbox"/>	MySQL	phpmyadmin	available	100.00%	47 Connections	None	db.t2.micro	vpc-e1c72988	Yes	master
<input type="checkbox"/>	MySQL	phpmyadmin-replica	available	99.51%	59 Connections	None	db.t2.micro	vpc-e1c72988	No	replica
<input type="checkbox"/>	MySQL	phpmyadmin2	available			None	db.t2.micro	vpc-e1c72988	No	replica

(b) Un maestro y un esclavo al máximo de utilización de CPU

	Engine	DB Instance	Status	CPU	Current Activity	Maintenance	Class
<input checked="" type="checkbox"/>	MySQL	phpmyadmin	available	100.00%	40 Connections	None	db.t2.micro
<input type="checkbox"/>	MySQL	phpmyadmin-replica	available	100.00%	50 Connections	None	db.t2.micro
<input type="checkbox"/>	MySQL	phpmyadmin2	available	100.00%	63 Connections	None	db.t2.micro

(c) Un maestro y dos esclavos al máximo de utilización de CPU

Figura 3.2: Estructura maestro esclavo

Obteniendo el siguiente resultado utilizando un maestro y dos esclavos:

```
Lifting the server siege...
Transactions:           4046 hits
Availability:           58.33 %
Elapsed time:           299.95 secs
Data transferred:       0.94 MB
Response time:          27.06 secs
Transaction rate:        13.49 trans/sec
Throughput:             0.00 MB/sec
Concurrency:            364.95
Successful transactions: 4046
Failed transactions:     2890
Longest transaction:    67.04
Shortest transaction:    0.00
```

Figura 3.3: Resultado final para un maestro y dos esclavos.

Se ha de mencionar que también se intento realizar un auto escalado de puertos.

4. Conclusión

A lo largo de este documento hemos podido observar como funciona Amazon RDS y su capacidad para escalar según sea la carga de trabajo mediante el uso de réplicas de lectura siguiendo la estructura maestro esclavo. También se ha podido observar como el manejo de este tipo de servicios es bastante complejo, aún así estamos bastante satisfechos ya que como mencionan en la página oficial de Amazon Web Services “Aunque los motores de base de datos relacionales ofrecen características y una funcionalidad potente, escalar una carga de trabajo más allá de una única instancia es una tarea bastante compleja y exige mucho tiempo y experiencia”.

Por último y tras la investigación realizada al encontrar las problemáticas a las que conducía la puesta en marcha de nuestro sistema AWS hemos realizado un análisis gracias al cuál vamos a sugerir que sistema montaríamos si se nos presentase un problema semejante al aquí mostrado.

Posibles mejoras del sistema:

- Utilizar Amazon Aurora en vez del motor de BD MySQL, ya que proporciona 5 veces el desempeño de MySQL.
- Utilizar Amazon ElastiCache en vez de Amazon EC2 porque al final encontramos en el apartado de preguntas frecuentes de ElastiCache que Amazon ElastiCache es un front-end ideal para datastores como Amazon RDS, proporcionando un nivel intermedio de alto desempeño para aplicaciones con tasas de solicitudes muy altas o bajos requisitos de latencia.

Referencias

- [1] Amazon Elastic Compute Cloud (Amazon EC2)
- [2] Auto Scaling
- [3] Amazon ElastiCache
- [4] Preguntas Frecuentes Amazon ElastiCache
- [5] Amazon Relational Database Service (Amazon RDS)
- [6] Amazon DynamoDB
- [7] Preguntas Frecuentes Amazon DynamoDB
- [8] Amazon Aurora