

Universidad de Alcalá de Henares - EPS

PL3 - Cloud Computing

Pablo Acereda David E. Craciunescu Ángel Martín
Ángelo Moreno Laura Pérez

May 26, 2019

1 Contenedores

2 IA & Machine Learning

3 IoT

Partiendo de una definición base, IoT (Internet of Things) es el conjunto de dispositivos que forman una red, la cual permite el intercambio de información y datos entre terminales.

Una vez que se tiene una cierta noción de lo que es IoT, es hora de pasar al desarrollo llevado a cabo en las dos plataformas empleadas: Microsoft Azure y Amazon Web Service.

Como última adenda a esta cuestión, decir que van a incluirse capturas de los procesos llevados a cabo, y para demostrar la realización del trabajo.

4 Seguridad

5 Web & Logic Apps

La parte de Web Logic Apps tiene un amplio espectro de trabajo, podemos realizar múltiples aplicaciones y servicios que aporten una perspectiva interesante y que permitan interactuar con ellas a un nivel satisfactorio.

De entre todas ellas, se ha elegido realizar una aplicación web con conexión remota. Será implementada tanto en Microsoft Azure como en Amazon Web Services, y posteriormente evaluada en función de la plataforma.

5.1 Prerrequisitos

Para realizar todo esto, primero se debe disponer de ciertas aplicaciones y recursos. Comenzando con la creación de las máquinas virtuales (en este caso con

distribución Linux) sobre las que se va a desarrollar la aplicación, tanto en Azure como en AWS.

En Azure, la creación de la máquina virtual será similar a la desarrollada en la parte obligatoria, pero se van a añadir unos requisitos que serán necesarios posteriormente. Estos son: creación de una IP estática y abrir los puertos 3000 y 5000. La IP estática será necesaria para mantener un contacto directo entre el servidor y la aplicación, y evitar cambiar continuamente la IP, con los problemas que puede acarrear. Abrir los puertos 3000 y 5000 se lleva a cabo para que la comunicación esté identificada en todo momento y se sepa a qué puerto llevar la información.

Figure 1: Selección de IP estática.

Tras esto, se realizará lo mismo en AWS, pero con las variaciones que la plataforma requiere. Primero, hay que crear la instancia de Linux que se va a utilizar. En este caso se utilizarán las características mínimas para no consumir demasiados recursos.

Básica

* Origen ⓘ
Any

* Intervalos de puertos de origen ⓘ
*

* Destino ⓘ
Any

* Intervalos de puertos de destino ⓘ
5000

* Protocolo
Any TCP UDP

* Acción
Permitir Denegar

* Prioridad ⓘ
1010

* Nombre
Port_5000


Descripción


Figure 2: Abrir puerto 5000.

Después, al igual que con Azure, se abrirán los puertos 3000 y 5000, por lo explicado anteriormente; y se crea una IP elástica, que te proporciona el propio AWS.

Una vez realizado esto, la plataforma pide que se descarguen un par de claves privadas para la conexión con la instancia, por lo tanto, se descarga el archivo con las claves y se ubica en algún directorio que se pueda recordar fácilmente, dado que más tarde será necesario.

Cuando se tienen listas ambas máquinas virtuales, hay que realizar la conexión ssh mediante la terminal para conectarse a ellas. En Azure, mediante el usuario, la IP estática y la contraseña, y en AWS mediante el usuario, la IP estática y la ruta

 **Agregar regla de seguridad de entrada** WebServerAPA-nsg ✕

 **Básica**

* Origen ⓘ

Any

▼

* Intervalos de puertos de origen ⓘ

*

* Destino ⓘ

Any

▼

* Intervalos de puertos de destino ⓘ

3000

✓

* Protocolo

Any

TCP

UDP

* Acción

Permitir

Denegar

* Prioridad ⓘ

1020

* Nombre

Port_3000

✓

Descripción

Figure 3: Abrir puerto 3000.

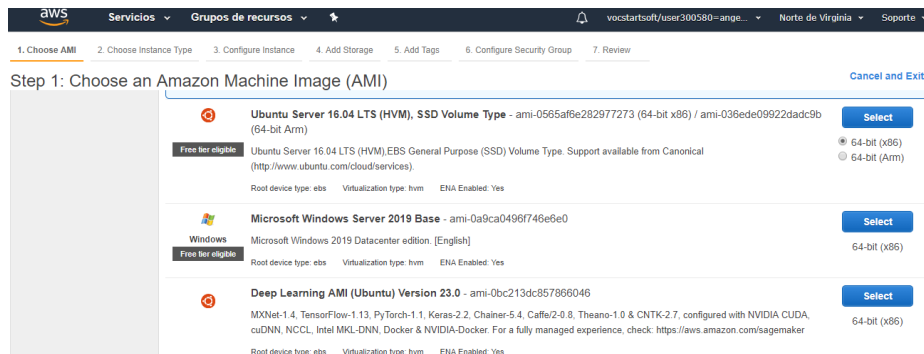


Figure 4: Selección de MV.

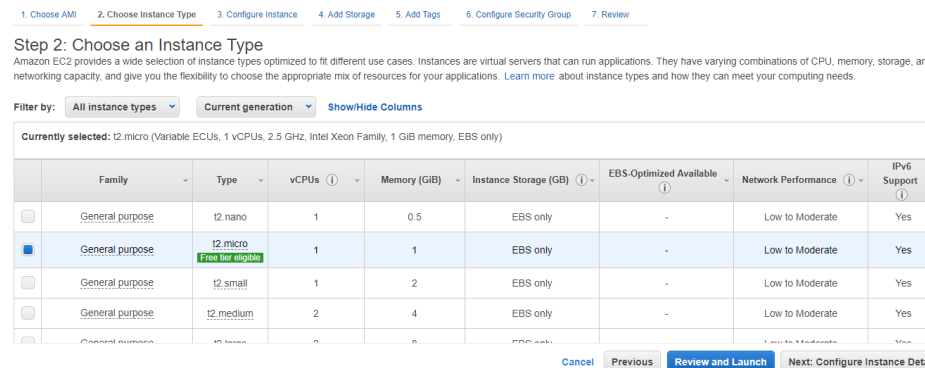


Figure 5: Elección de características.

donde se ha ubicado el archivo con las claves privadas.

Ya dentro de las máquinas virtuales, se instalarán Python 3, NodeJS y npm, tanto en la de Azure como en la de AWS.

5.2 Funcionamiento

Para el desarrollo y la implementación de la aplicación, se comenzará con la creación del código necesario para ella. En este caso, se ha recurrido tanto a Python 3 como a React, dadas las amplias opciones que nos proporcionan y lo sencillo que resulta desarrollar cualquier cosa con estos lenguajes. El código se expone a continuación:

En sí, lo que se está realizando es una página web con un texto, un contador y un botón. El botón será para ir incrementando el contador de forma remota y que sea registrado.

1. Choose AMI 2. Choose Instance Type 3. Configure Instance 4. Add Storage 5. Add Tags 6. Configure Security Group 7. Review

Step 6: Configure Security Group

A security group to a set of instances rules that control the traffic to your instance. On this page, you can add rules to allow specific traffic to reach your instance. For example, if you want to set up a x allow Internet traffic to reach your instance, add rules that allow unrestricted access to the HTTP and HTTPS ports. You can create a new security group or select from an existing one below. [Learn](#) Amazon EC2 security groups.

Assign a security group: ☒ Create a new security group ☐ Select an existing security group

Security group name:

Description:

Type	Protocol	Port Range	Source	Description
Custom TCP	TCP	3000	Custom CIDR, IP or Security Group	e.g. SSH for Admin Desktop
Custom TCP	TCP	5000	Custom CIDR, IP or Security Group	e.g. SSH for Admin Desktop

[Add Rule](#)

Warning

You will not be able to connect to this instance as the AMI requires port(s) 22 to be open in order to have access. Your current security group doesn't have port(s) 22 open.

Figure 6: Abrir puertos.

Para comprobar el funcionamiento de la aplicación, se abren dos terminales para la plataforma sobre la que vayamos a trabajar, se lleva a cabo la conexión ssh con lo mencionado anteriormente y, dado que el código ha sido alojado en repositorios de Git, se clonan ambos repositorios en el directorio de trabajo que se desee. Una vez hecho, dentro del repositorio donde está la parte de la web, se ejecuta el comando “npm start”, para iniciar la conexión de la página web. Como los pasos a seguir son los mismos en ambas plataformas, se va a mostrar lo de una de ellas (aplicable a la otra por igual).

Mientras, en la otra terminal, para iniciar el servidor llamamos al comando "FLASK APP=main.py flask run --host=0.0.0.0".

Con esto se tiene tanto la web activa, como el servidor para realizar los cambios iniciado. Para comprobarlo se accede a la url de la web “ip-estática:3000”, y se pulsa varias veces el botón del contador.

Se puede observar que el contador aumenta, y si se comprueba la terminal en la que se ha iniciado el servidor, aparecen las peticiones que ha recibido y el estado en cada momento de dicho contador.

5.3 Comparación

Por último, como comparación entre ambas plataformas, a la hora de crear e iniciar los recursos, Azure ofrece más facilidades y más variedad que AWS. Azure tiene mucha más variedad de entornos y utilidades para empezar a desarrollar, mientras que AWS lo tiene más limitado y el acceso está restringido a algunos de los recursos (por lo menos para la cuenta de estudiantes). Pero también hay que destacar que AWS ofrece unos protocolos de seguridad más estrictos y a la hora de iniciar las conexiones y utilizar los recursos es más veloz que Azure.

Por lo tanto, si se quiere utilizar en un ámbito menos profesional y sin llevar a cabo grandes proyectos, se recomienda el uso de Microsoft Azure. Pero si se pretende hacer un proyecto empresarial a mayor escala, y se disponen de los

```

1  from flask import Flask
2  from flask import jsonify
3  from flask import request
4  from flask_cors import CORS
5
6  server_counter = "0"
7
8  app = Flask(__name__)
9  CORS(app)
10 @app.route("/")
11 def hello():
12     return "%s" % server_counter
13
14 @app.route("/set/<counter>", methods=['PUT'])
15 def setCounter(counter):
16     global server_counter
17     server_counter = counter
18     print(server_counter)
19     return '200'

```

Figure 7: Código para el Servidor.

recursos y el personal adecuados, se recomienda la utilización de Amazon Web Services.


```

1 import * as React from 'react';
2 import Counter from './Counter';
3 import axios from 'axios'
4 import './App.css';
5
6
7 class App extends React.Component {
8
9   constructor(props){
10     super(props)
11     this.state = {
12       counter: 1
13     }
14     this.sendToAPI = this.sendToAPI.bind(this)
15   }
16
17   componentDidMount(){
18     axios.get('http://localhost:5000')
19     .then(response => {
20       this.setState({
21         counter: response.data})
22     })
23     .catch(
24       err => console.log(err)
25     )
26   }
27
28   sendToAPI(counter){
29     axios.put('http://localhost:5000/set/'+ counter)
30     .then(
31       response => console.log(response)
32     )
33     .catch(
34       err => console.log(err)
35     )
36   }
37
38   render() {
39     return (
40       <div className="App">
41         <header className="App-header">
42           <Counter counter={this.state.counter} />
43         </div>
44         <button className="Addbutton" onClick={() => {
45           const newCounter = this.state.counter + 1
46           this.setState({
47             counter: newCounter
48           })
49           this.sendToAPI(newCounter)
50         }}>
51           Hello</button>
52       </div>
53     </header>
54   </div>
55   );
56 }
57
58 export default App;

```

Figure 8: Código para la Web.

```
angel@WebServerAPA:~/web$ cd webAngel/  
angel@WebServerAPA:~/web/webAngel$ npm start  
  
> global-counter@0.1.0 start /home/angel/web/webAngel  
> react-scripts start
```

Figure 9: Conexión de la web.

```
# FLASK_APP=main.py flask run --host=0.0.0.0  
angel@WebServerAPA:~/web/AngelServer$ FLASK_APP=main.py flask run --host=0.0.0.0  
* Serving Flask app "main.py"  
* Environment: production  
  WARNING: This is a development server. Do not use it in a production deployment.  
  Use a production WSGI server instead.  
* Debug mode: off  
* Running on http://0.0.0.0:5000/ (Press CTRL+C to quit)
```

Figure 10: Conexión del Servidor.

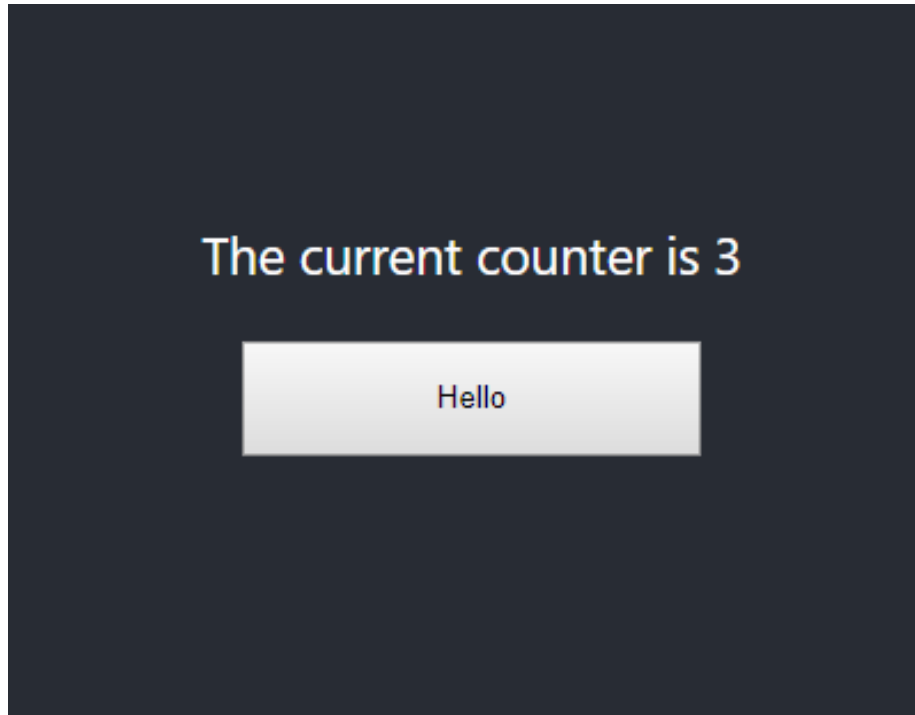


Figure 11: Contador en la web.

```
* Running on http://0.0.0.0:5000/ (Press CTRL+C to quit)
83.43.15.56 - - [26/May/2019 20:04:06] "GET / HTTP/1.1" 200 -
83.43.15.56 - - [26/May/2019 20:04:26] "OPTIONS /set/1 HTTP/1.1" 200 -
1
83.43.15.56 - - [26/May/2019 20:04:26] "PUT /set/1 HTTP/1.1" 200 -
83.43.15.56 - - [26/May/2019 20:04:26] "OPTIONS /set/2 HTTP/1.1" 200 -
2
83.43.15.56 - - [26/May/2019 20:04:26] "PUT /set/2 HTTP/1.1" 200 -
83.43.15.56 - - [26/May/2019 20:04:26] "OPTIONS /set/3 HTTP/1.1" 200 -
3
83.43.15.56 - - [26/May/2019 20:04:26] "PUT /set/3 HTTP/1.1" 200 -
```

Figure 12: Comprobación en el Servidor.