Samuel I. Magdaleno Álvarez
18416

Tarea 1

Explicar que significa "Microservice Architecture":

La arquitectura basada en microservicios es un método de desarrollo de software que estructura una aplicación como una colección de servicios pequeños, que se ejecutan en su propio ambiente y se comunican entre si a través de diferentes protocolos. Entre los más comunes protocolos de comunicación están HTTP/REST con JSON.

Aplicaciones construidas a base de micro servicios ponen mucho énfasis en monitoreo en tiempo real, ya que los servicios pueden fallar en cualquier momento, es importante detectar y reparar fallas rápidamente. Logran esto a base de la revisión de elementos arquitectónicos (cuantos request por segundo se están mandando a la base de datos) y métricas relevantes para negocios (como cuantas ordenes están entrando por minuto).

El completo opuesto de la arquitectura en microservicios es la monolítica, ya que esta está construida como una sola unidad autónoma.

Algunos ejemplos de aplicación que actualmente usan esta arquitectura son: Netflix, eBay, Amazon, Twitter, PayPal, Soundcloud, etc.

Bash para crear el ambiente de esta clase:

```
#!/bin/bash

echo "Installing aws cli"

sudo apt-get install awscli

echo "this is the aws version: "

aws --version

sudo apt-get install python-software-properties

curl -sL https://deb.nodesource.com/setup_8.x | sudo -E bash -

echo "Installing nodejs"

sudo apt-get install nodejs

echo "this is the node version:"

node -v

echo "this is the npm version:"

npm -v

sudo apt-get update

echo "Installing yeoman"

sudo npm install -g yo
```

Samuel I. Magdaleno Álvarez
18416

```
echo "Installing grunt"

sudo npm install -g grunt-cli

echo "this is the grunt version:"

echo --version

echo "Installing gulp"

sudo npm install -g gulp

echo "this is the gulp version:"

gulp --version

echo "amiga o amigo, tu ambiente está listo para la clase de cloud computing (/* 3 *)/"
```

Bash para hacerle deploy a mi página:

```
#!/bin/bash

COMMIT="$1"

echo "Regresando al folder principal"

cd

echo "Entrando a samweb"

cd samweb

echo "Grunteando la página"

grunt build

cd dist

echo "creando y sincronizando con bucket"

aws s3api create-bucket --bucket samweb

aws s3 sync . s3://samweb --acl public-read

echo "agregando permisos"

echo '{"IndexDocument":{"Suffix":"index.html"},"ErrorDocument":{"Key":"error.html"}}' | sudo tee perm.json

aws s3api put-bucket-website --bucket samweb --website-configuration file://perm.json

echo "Commiteando cambios"

cd ..
```

Samuel I. Magdaleno Álvarez
18416

```
git add .

git commit -am "$COMMIT"

echo "Pushando cambios"

git push

echo "este fue tu commit: $COMMIT"
```

NOTA! Para esta tarea primero pensé que se trataba de un bash que hacia deploy a un website, independientemente de que fuera o no mi página, entonces hice un bash que recibe como parámetro el nombre del website que quiere ser creado y ya. Crea un website y le hace deploy sin otro input del usuario más que enters. Es lo que más tarde en la tarea y pues, no era necesario, igual lo anexo por si quiere darme algún punto extra *hint* *hint*.

https://docs.google.com/document/d/1kzGQMFPFU_XFd9aEoBu1SHAJvW47Af4_dIHaUH7U2lM/edit?usp=sharing **(como link de drive, para no llenar más espacio)**

Explicar cómo funcionan las rutas en angular, y poner un ejemplo de código:

Las rutas en angular ayudan a enlazar la vista y los controladores. Se usan principalmente para crear single page applications, lo que significa que para cambiar entre áreas de la página o tabs, no es necesario recargar el navegador para que el contenido cambie o aparezca. El módulo de angular que hace todo esto posible se llama ngRoute.

ngRoute usa el servicio de $templateCache, lo que hace que la plantilla se carge por completo y almacene todos los componentes en la memoria cache y pueda ser utilizada desde está en los cambios de página.

ngRoute se declara en el controlador y hace uso del $routeProvider que lee los cambios en el url y asigna las vistas que van a ser desplegadas. Este espacio se asigna en la página por medio de ng-view, que reserva el espacio en el html y es donde se despliegan los bloques de código asignados por el controlador.

Ejemplo simple de w3school:

```
<body ng-app="myApp">

<p><a href="#/!">Main</a></p>

<a href="#!red">Red</a>
<a href="#!green">Green</a>
<a href="#!blue">Blue</a>

<div ng-view></div>

<script>
var app = angular.module("myApp", ["ngRoute"]);
app.config(function($routeProvider) {
   $routeProvider
```

Samuel I. Magdaleno Álvarez
18416

```
    .when("/", {
        templateUrl : "main.htm"
    })
    .when("/red", {
        templateUrl : "red.htm"
    })
    .when("/green", {
        templateUrl : "green.htm"
    })
    .when("/blue", {
        templateUrl : "blue.htm"
    });
});
</script>
</body>
```

Como tener más de un access Key en tu computadora y ejemplo en código:

Los usuarios necesitan sus propias claves de acceso para realizar llamadas programáticas a AWS desde la Interfaz de línea de comandos de AWS (AWS CLI), Herramientas para Windows PowerShell, los SDK de AWS o llamadas HTTP directas utilizando las API para servicios AWS individuales. Cada usuario puede tener dos claves de acceso activas.

Se pueden crear llaves de acceso a través de la creación de otros perfiles en la misma máquina, lo que les asigna sus propias llaves de acceso a cada uno. El comando que se usa para esto es:

`aws configure –profile [cuenta nueva]`

Esto genera un nuevo archivo de credentials, en el que se muestra la información de dos perfiles:

`~/.aws/credentials`

```
[default]
aws_access_key_id=AKIAIOSFODNN7EXAMPLE
aws_secret_access_key=wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY

[user2]
aws_access_key_id=AKIAI44QH8DHBEXAMPLE
aws_secret_access_key=je7MtGbClwBF/2Zp9Utk/h3yCo8nvbEXAMPLEKEY
```

Posteriormente se tendrá que especificar que cuenta se usa en cada comando de AWS CLI.

Otra forma de crear llaves de acceso es con el comando “**`créate-access-key`**”, que genera nuevas AWS secret Access keys y el ID del AWS Access key para el usuario especificado.

Ejemplo:

Samuel I. Magdaleno Álvarez
18416

```
aws iam create-access-key --user-name Bob
```

Output:

```
{
    "AccessKey": {
        "UserName": "Bob",
        "Status": "Active",
        "CreateDate": "2015-03-09T18:39:23.411Z",
        "SecretAccessKey": "wJalrXUtnFEMI/K7MDENG/bPxRfiCYzEXAMPLEKEY",
        "AccessKeyId": "AKIAIOSFODNN7EXAMPLE"
    }
}
```

Si no se especifica un usuario, IAM lo determina basado en el ID del Access key que se usó en el request.

También se pueden crear y utilizar claves de acceso temporal, conocidas como credenciales de seguridad temporales. La ventaja de las credenciales de seguridad temporales es que son a corto plazo; tienen una fecha de expiración. Se puede utilizar claves de acceso temporal en entornos menos seguros o distribuirlas para conceder a los usuarios acceso temporal a los recursos de una cuenta de AWS.

¿Qué voy a hacer diferente este semestre, basado en la experiencia del anterior?

Primero que nada, este semestre le voy a dedicar más tiempo a hacer las tareas desde el principio. El semestre pasado hacia las tareas la mayoría de las veces desde el sábado, y es algo que tengo que cambiar (empezando por esta tarea [que voy bien porque la empecé el miércoles]). Otra cosa que tengo hacer es dedicarle más tiempo a los temas de la materia, no necesariamente crea que el semestre pasado no le dedique lo suficiente (creo que es una de las cosas buenas que hice el semestre pasado), sino porque los temas de esta clase son completamente nuevos para mí y se ve que son de un nivel de dificultad más elevado. En todo lo demás, considero que hice un buen trabajo el semestre pasado, voy a seguir haciendo las actividades extras, dedicarle el tiempo necesario al proyecto, entre otras cosas.

Ahora si voy a leer el libro este semestre…