INFRAESTRUCTURAS DE CLOUD PÚBLICO

dooros@posgrado.upv.es

Universidad Politécnica de Valencia

Mater Universitario en computación paralela y distribuida

Infraestructuras de Cloud Público

Despliegue de API REST en Amazon

Contenido

[Introducción 2](#_Toc506987278)

[Esquema General 3](#_Toc506987279)

[Objetivos 3](#_Toc506987280)

[Desarrollo 4](#_Toc506987281)

[Requisitos previos 4](#_Toc506987282)

[Otorgar permisos a nuestro usuario Linux 4](#_Toc506987283)

[Instalación de CLI Amazon 4](#_Toc506987284)

[Instalación de Ansible 4](#_Toc506987285)

[Scripts de configuración 5](#_Toc506987286)

[Ejecución 9](#_Toc506987287)

[Creación de Base de datos 9](#_Toc506987288)

[Creación de infraestructura de alta disponibilidad 11](#_Toc506987289)

[Conclusiones 15](#_Toc506987290)

[Referencias 16](#_Toc506987291)

# Introducción

El uso de tecnologías como es el Cloud Computing es cada vez más aceptado por las empresas, quienes ahora están más dispuestos a pagar por infraestructuras albergadas en la nube y con ello reducen los costos de implementación y manteamiento a gran escala por no decir casi cero. Este documento presenta una forma de desplegar una API REST en la plataforma de infraestructura pública como lo es Amazon Web Service con una configuración de auto escalado a una aplicación Stateless que nos va garantizar alta disponibilidad de nuestro servicio web. A lo largo del desarrollo se hará uso de los servicios de Amazon con ser su CLI, EC2 y la herramienta Ansible que nos servirá para crear una AMI (Amazon Machine Image) personalizada con nuestra aplicación.

# Esquema General

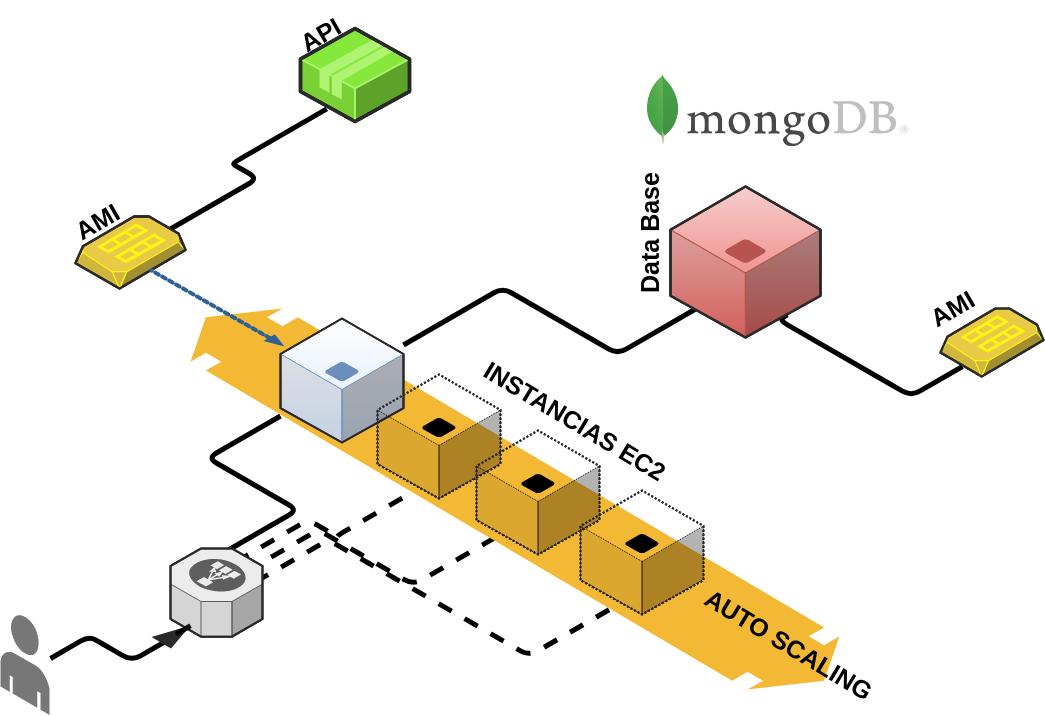
## Objetivos

– Utilizar Amazon EC2 para el despliegue de API REST.

– Incluir grupo de auto-escalado.

– Creación de AMI personalizada para la aplicación.

– Utilizar AWS CLI para automatizar el despliegue.



Nuestra infraestructura consta de una instancia EC2 configurada con un gestor de base de datos de MongoDB. Luego tendremos una AMI configurada con un servidor NodeJS y nuestra aplicación escuchando el puerto 8080. Luego crearemos un balanceador de carga y por último configuraremos un grupo de auto escalado a partir de nuestra AMI.

# Desarrollo

## Requisitos previos

Para realizar de forma automatizada el despliegue de nuestra API REST haremos tendremos como sistema operativo una distribución de Linux, en este caso se realizó con la distribución Linux Mint y con ella las herramientas CLI de Amazon y Ansible.

## Otorgar permisos a nuestro usuario Linux

Para una mejor experiencia en el uso del script otorgaremos a nuestro usuario todos los permisos mediante los siguientes comandos:

sudo visudo

# User privilege specification

root ALL=(ALL:ALL) ALL

miusuario ALL=(ALL:ALL) ALL

## Instalación de CLI Amazon

sudo apt-get install python-pip

pip install awscli

Una vez instalado en nuestro sistema realizaremos la configuración de acceso a la plataforma de Amazon mediante el comando:

aws configure

AWS Access Key ID [None]: XXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

AWS Secret Access Key [None]: XXXXXXXXXXXXXXXXXXXXXXXXXX

Default region name [None]: us-west-2

Default output format [None]: json

Tendremos que agregar nuestras credenciales como también la región con la que estamos registrados.

## Instalación de Ansible

Ansible es una herramienta de software libre para la configuración y administración de computadoras. Para instalar la herramienta ejecutaremos los siguientes comandos:

sudo apt-get install software-properties-common

sudo apt-add-repository ppa:ansible/ansible

sudo apt-get update

sudo apt-get install ansible

Para tener una mejor experiencia en el uso de ansible desactivaremos la aprobación de host key en el archivo de configuración de ansible.

sudo nano /etc/ansible/ansible.cfg

host\_key\_checking = False

## Scripts de configuración

Creación de instancia con base de datos:

***create\_database.sh***

#!/bin/bash

id=$(aws ec2 run-instances --image-id ami-841f46ff --key-name alucloud35-keypair --security-group-ids sg-c171d4b4 --instance-type t2.micro --subnet-id subnet-2bfb6c4f --query 'Instances[\*].[InstanceId]' --output text)

echo $id

while true; do

code=$(aws ec2 describe-instance-status --instance-ids $id --query 'InstanceStatuses[\*].InstanceStatus.[Status]' --output text)

echo $code

if [ "$code" = "ok" ]; then

dns=$(aws ec2 describe-instances --instance-ids $id --query 'Reservations[\*].Instances[\*].[PublicDnsName]' --output text)

echo $dns

break

fi

sleep 8

done

sudo echo -e "[webserver]\n$dns">/etc/ansible/hosts

sudo echo -e "module.exports={IP\_DATABASE:'$dns'}">config.js

ansible-playbook -e 'host\_key\_checking=False' --private-key alucloud35-keypair.pem mongodb.yml

Las letras en rojo especifican recursos previos creados en plataforma de Amazon que no se explican en este documento, pero algunos archivos se incluyen en la carpeta del proyecto. Los pasos que sigue el script anterior son:

1. Creación de una instancia a partir de una AMI Linux en el servicio EC2 de Amazon.
2. Verificación del estado de creación de la instancia.
3. Configuración del archivo host de ansible (se necesitan permisos de root para ejecutar este comando).
4. Copiamos el DNS de nuestra instancia de base de datos al archivo de configuración de nuestra aplicación.
5. Ejecución de la herramienta ansible para la instalación y configuración de nuestro gestor de base de datos.

A continuación, se muestra el archivo de confirmación usado por ansible para instalar MongoDB como gestor de base de datos no relacional.

---

- hosts: all

user: ubuntu

gather\_facts: True

sudo: True

tasks:

- name: Update and upgrade apt packages

become: true

apt:

upgrade: yes

- name: MongoDB | Import public key

apt\_key:

keyserver: hkp://keyserver.ubuntu.com:80

id: EA312927

- name: MongoDB | Add repository

apt\_repository:

filename: '/etc/apt/sources.list.d/mongodb-org-3.2.list'

repo: 'deb http://repo.mongodb.org/apt/ubuntu trusty/mongodb-org/3.2 multiverse'

state: present

update\_cache: yes

- name: MongoDB | Install MongoDB

sudo: yes

apt:

force: yes

name: mongodb-org

state: present

update\_cache: yes

- name: creating directory

shell: cd / && sudo mkdir data && cd data && sudo mkdir db

- copy:

src: mongod.conf

dest: /etc/mongod.conf

- copy:

src: mongodb.service

dest: /etc/systemd/system/mongodb.service

- name: install systemd

become: yes

apt: name=systemd

- name: start mongodb

shell: sudo systemctl start mongodb

- name: service enable mongodb

shell: sudo systemctl enable mongodb

- name: reset service mongod

shell: sudo service mongod restart

Creación de AMI y Auto escalado de aplicación:

***create\_app.sh***

#!/bin/bash

id=$(aws ec2 run-instances --image-id ami-841f46ff --key-name alucloud35-keypair --security-group-ids sg-c171d4b4 --instance-type t2.micro --subnet-id subnet-2bfb6c4f --query 'Instances[\*].[InstanceId]' --output text)

echo $id

while true; do

code=$(aws ec2 describe-instance-status --instance-ids $id --query 'InstanceStatuses[\*].InstanceStatus.[Status]' --output text)

echo $code

if [ "$code" = "ok" ]; then

dns=$(aws ec2 describe-instances --instance-ids $id --query 'Reservations[\*].Instances[\*].[PublicDnsName]' --output text)

echo $dns

break

fi

sleep 8

done

sudo echo -e "[webserver]\n$dns">/etc/ansible/hosts

echo "Configurando AMI"

ansible-playbook -e 'host\_key\_checking=False' --private-key alucloud35-keypair.pem node.yml

ami=$(aws ec2 create-image --instance-id $id --name "ICP\_AMI\_API\_$id" --description "API nodejs" --output text)

echo $ami

while true; do

code=$(aws ec2 describe-images --image-ids $ami --query 'Images[\*].[State]' --output text)

echo "estado de AMI: "$code

if [ "$code" = "available" ]; then

break

fi

sleep 5

done

echo "ami creada!!"

echo "creando configuracion de autoscaling.."

aws autoscaling create-launch-configuration --launch-configuration-name launch-config-$id --image-id $ami --instance-type t2.micro --security-groups sg-c171d4b4 --key-name alucloud35-keypair --associate-public-ip-address

echo "creando balanceador de carga.."

aws elbv2 create-load-balancer --name lb-$id --subnets subnet-2bfb6c4f subnet-c2f25afd --scheme internet-facing --security-groups sg-c171d4b4

lb=$(aws elbv2 describe-load-balancers --names lb-$id --query 'LoadBalancers[\*].[LoadBalancerArn]' --output text)

echo "creando target group..."

tg=$(aws elbv2 create-target-group --name lb-$id-tg --protocol HTTP --port 8080 --vpc-id vpc-83a213fb --query 'TargetGroups[\*].[TargetGroupArn]' --output text)

#registramos las isntancias

echo "Register target group..."

aws elbv2 register-targets --target-group-arn $tg --targets Id=$id

#asignamos tg a lb

echo "creando listener de balanceador de carga..."

aws elbv2 create-listener --load-balancer-arn "$lb" --protocol HTTP --port 8080 --default-actions Type=forward,TargetGroupArn="$tg"

echo "creando grupo de autoescalado..."

aws autoscaling create-auto-scaling-group --auto-scaling-group-name as-group-$id --launch-configuration-name launch-config-$id --min-size 1 --max-size 2 --default-cooldown 120 --target-group-arns $tg --vpc-zone-identifier "subnet-2bfb6c4f,subnet-c2f25afd"

link=$(aws elbv2 describe-load-balancers --names lb-$id --query 'LoadBalancers[\*].[DNSName]' --output text)

echo "conectar a: www.$link:8080"

El script anterior sigue los siguientes pasos:

1. Creación de una instancia a partir de una AMI Linux en el servicio EC2 de Amazon.
2. Verificación del estado de creación de la instancia.
3. Configuración del archivo host de ansible (se necesitan permisos de root para ejecutar este comando).
4. Ejecución de la herramienta ansible para la instalación y configuración de nuestra aplicación API REST.
5. Creación de nuestra AMI personalizada que incluye un servidor NodeJS y nuestra aplicación en el directorio **/home/Ubuntu/proyect**
6. Verificación del estado de la creación de la AMI.
7. Creación de la configuración del grupo de auto escalado que hace uso de nuestra AMI personalizada.
8. Creación del balanceador de carga.
9. Creación de Target Group con el puerto 8080.
10. Registramos en nuestro Target Group nuestra instancia creada.
11. Creamos un listener para nuestro balanceador de carga que escuchara el puerto 8080.
12. Creación del grupo de auto escalado con un mínimo de una instancia y máximo de dos.

# Ejecución

Para crear nuestras instancias como también nuestro grupo de auto escalado tendremos que estar en carpeta proyect que puedes descargar del siguiente repositorio de GitHub: <https://github.com/adolfo24/autoscaling_aws>

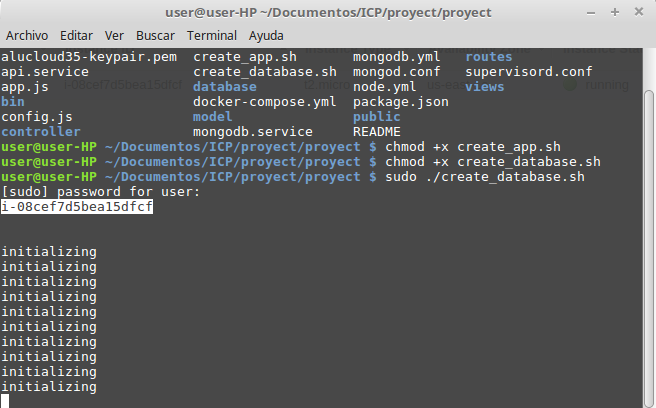
cd autoscaling\_aws/proyect

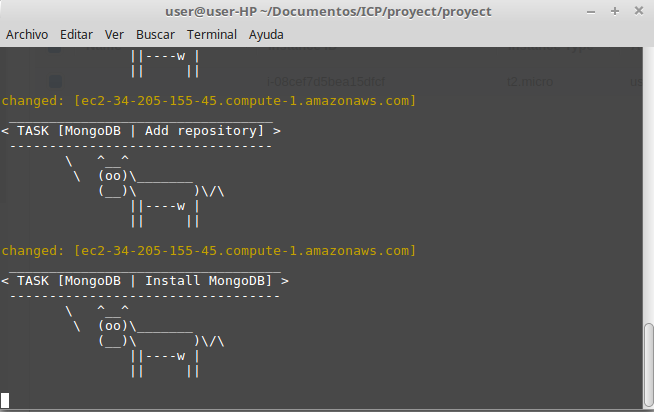
## Creación de Base de datos

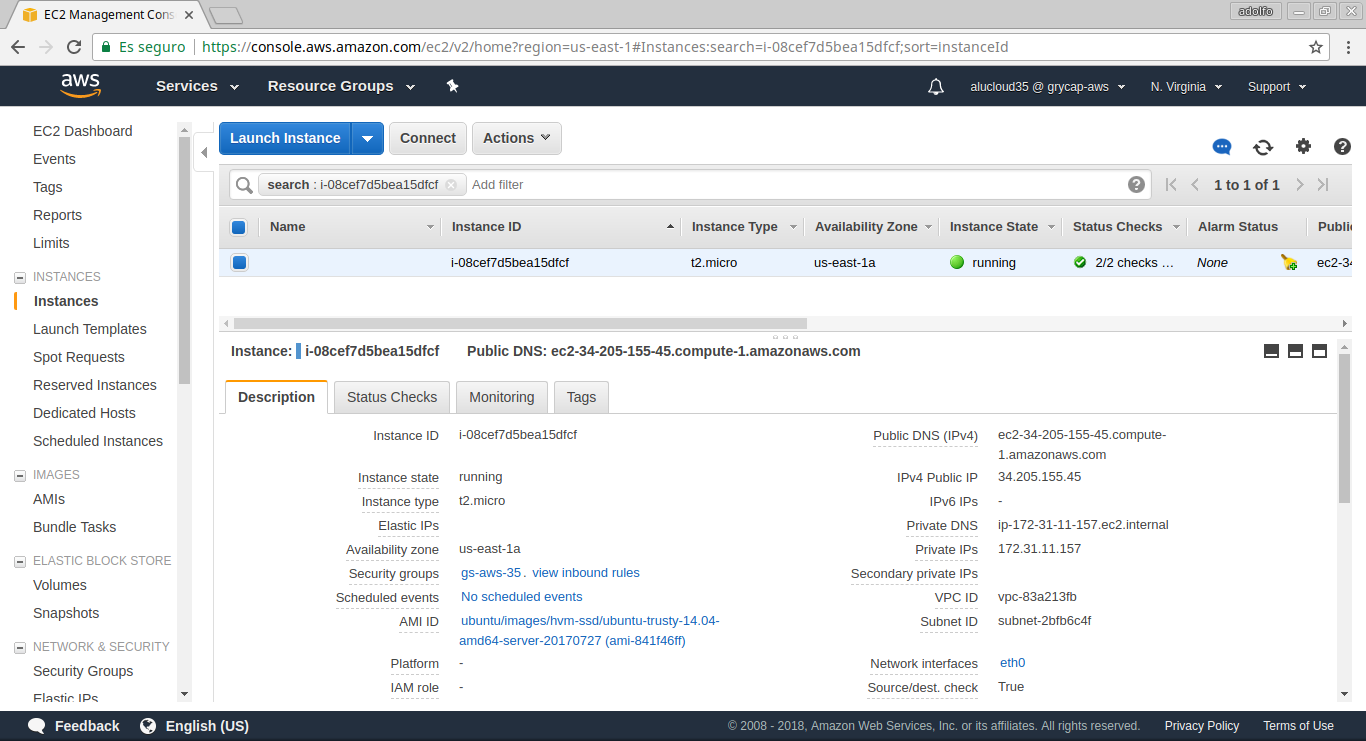
sudo chmod +x create\_database.sh

sudo ./create\_database.sh

Con esto se lanzará automáticamente una instancia en el servicio EC2 de Amazon con MongoDB como gestor de base de datos para nuestra aplicación.





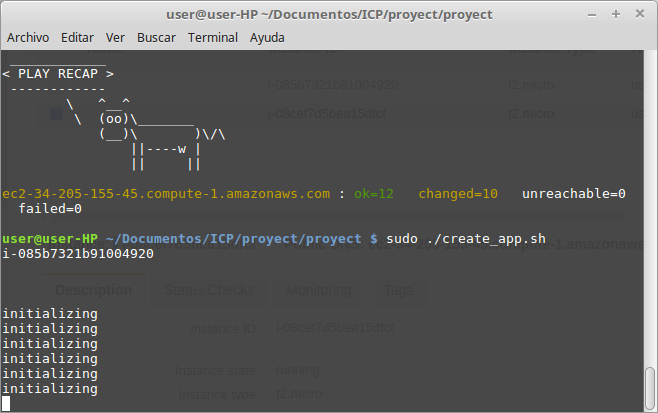


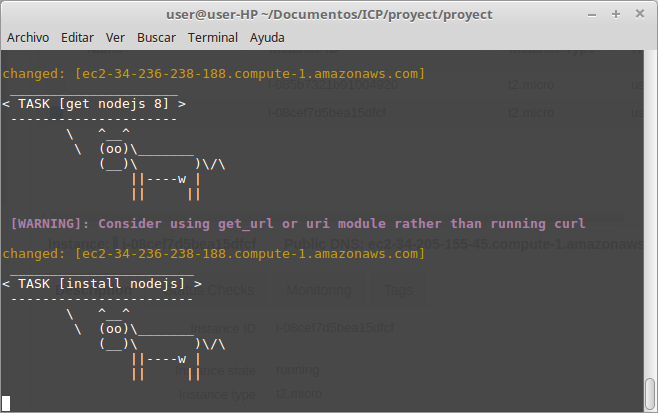
## Creación de infraestructura de alta disponibilidad

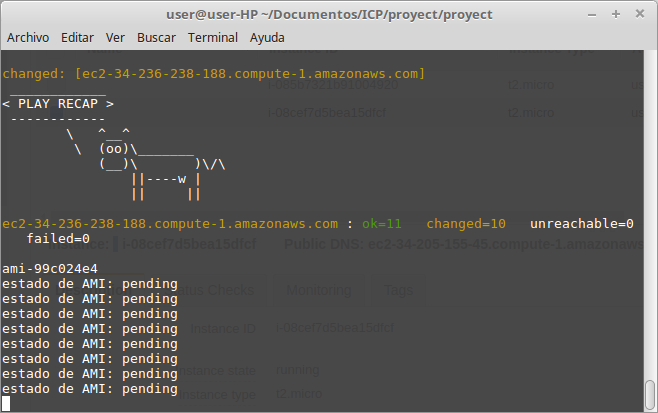
Ubicados en el mismo directorio ejecutamos los siguientes comandos para lanzar en los servicios de Amazon nuestra infraestructura de alta disponibilidad.

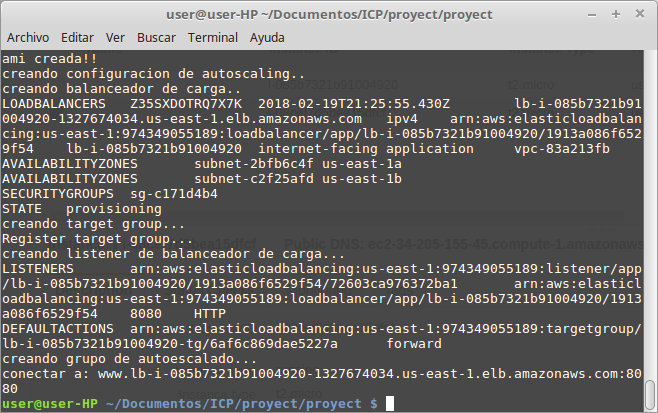
sudo chmod +x create\_app.sh

sudo ./create\_app.sh



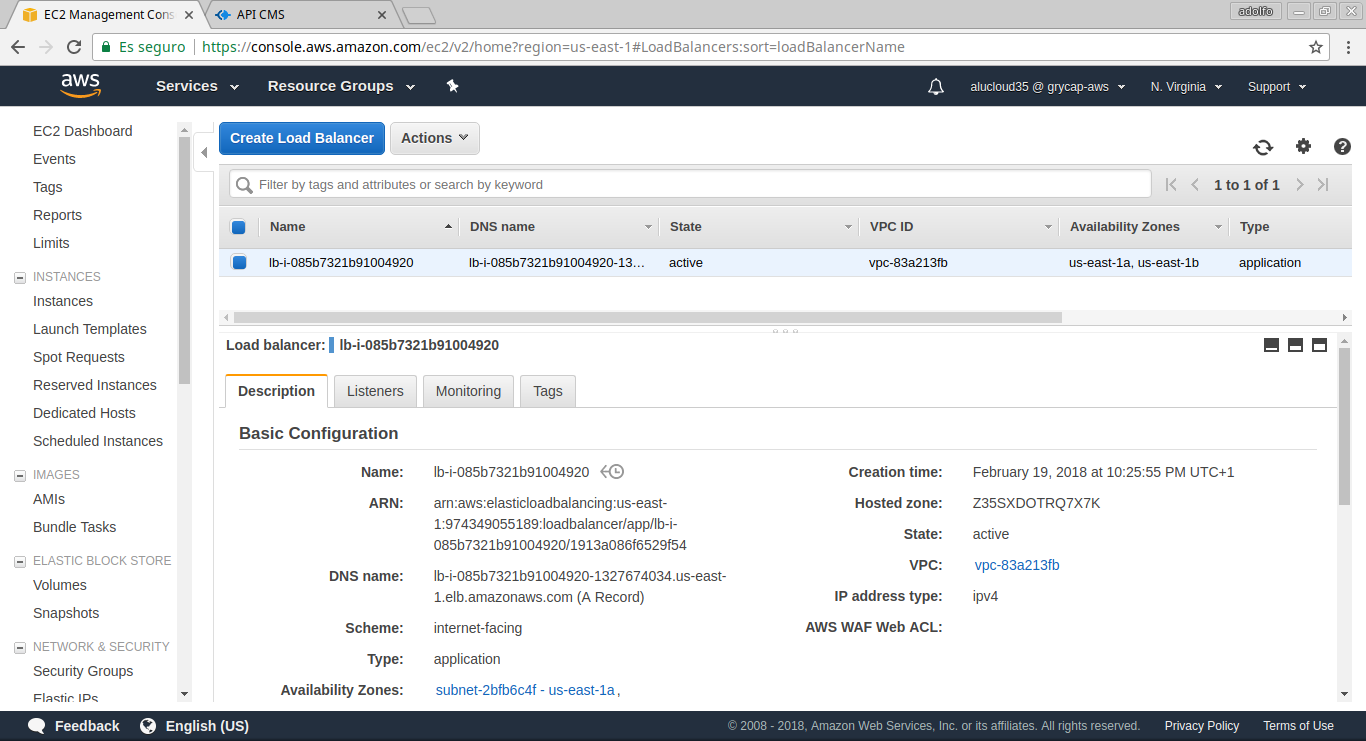




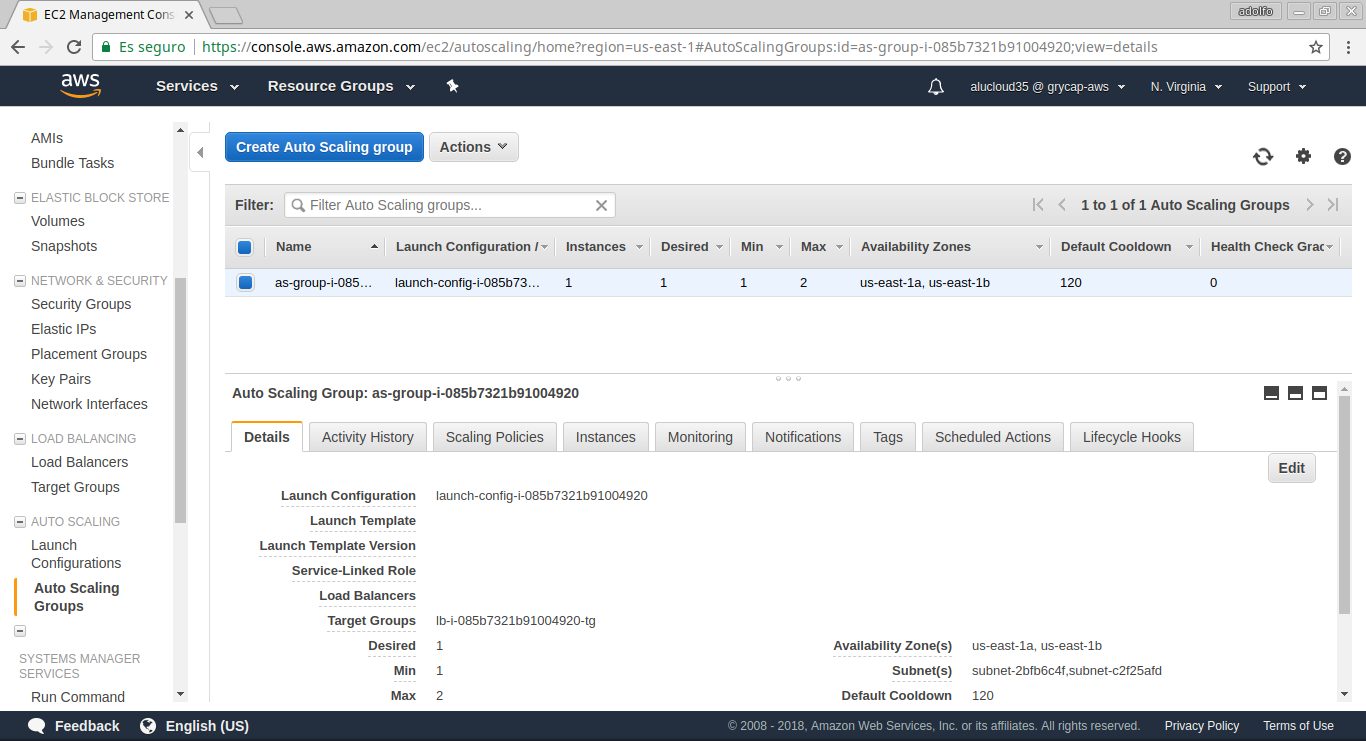


al finalizar la ejecución de este script contaremos con:

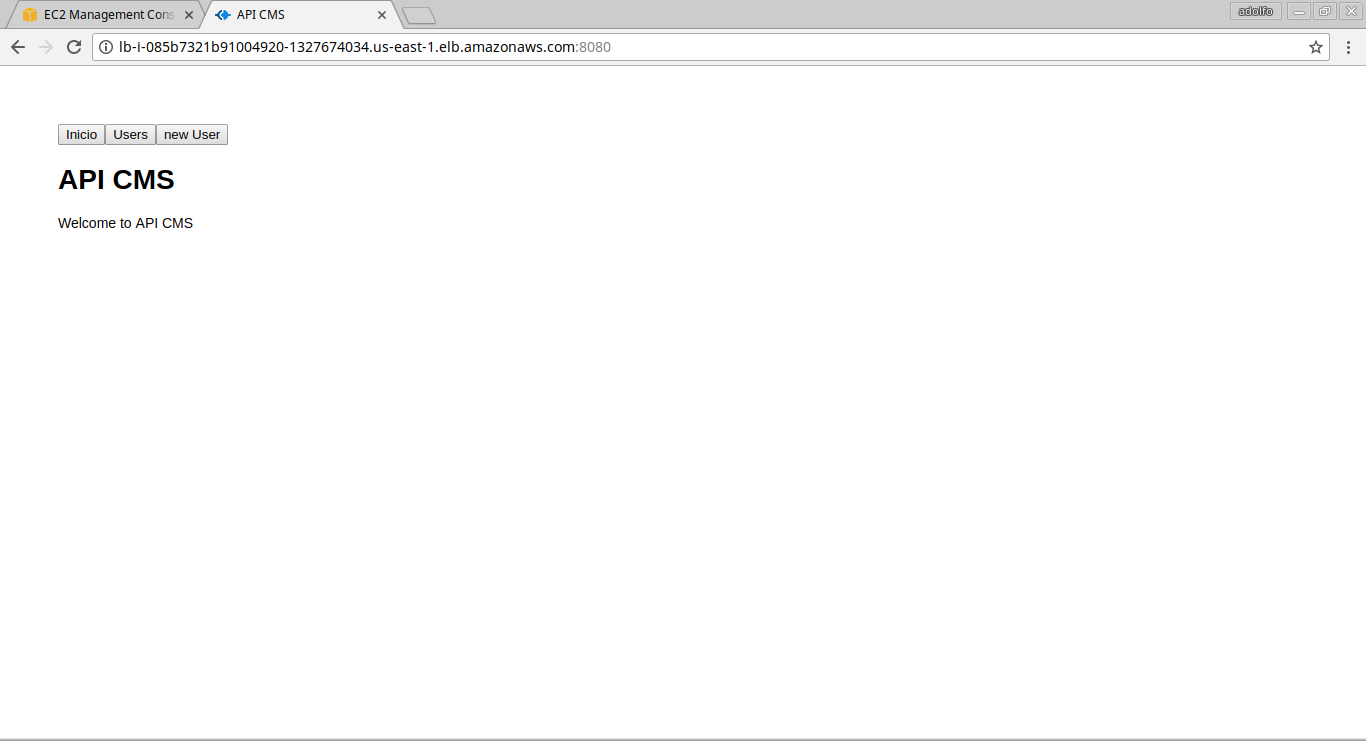
1. Una instancia EC2 sobre la que corre nuestra aplicación escuchando en el puerto 8080.
2. Un balanceador de carga que sirve nuestra aplicación en el puerto 8080.



1. Un grupo de auto escalado que tendrá siempre como mínimo una instancia y máximo dos para mantener nuestro servicio siempre disponible.



1. Nuestra aplicación en la dirección DNS de nuestro balanceador de carga escuchando el puerto 8080.



# Conclusiones

1. El uso de plataformas que brindan servicios de infraestructura en la nube actualmente está siendo cada vez más aceptado y usado tanto por la empresa privada como por entidades gubernamentales gracias a su facilidad de implementación y bajos costes.
2. Si somos emprendedores y queremos tener nuestros servicios en internet, las plataformas cloud hasta el momento es la mejor opción para echar a andar nuestros servicios a un bajo coste y poder escalar a medida que nuestro servicio sea más exigido.
3. Amazon Web Service por hoy es una de las plataformas de infraestructuras más populares que cuenta con una amplia gama de servicios con los que podemos hacer infraestructuras de para la mayoría de aplicaciones.

# Referencias

1. <https://aws.amazon.com/es/documentation/>
2. <http://docs.ansible.com/>