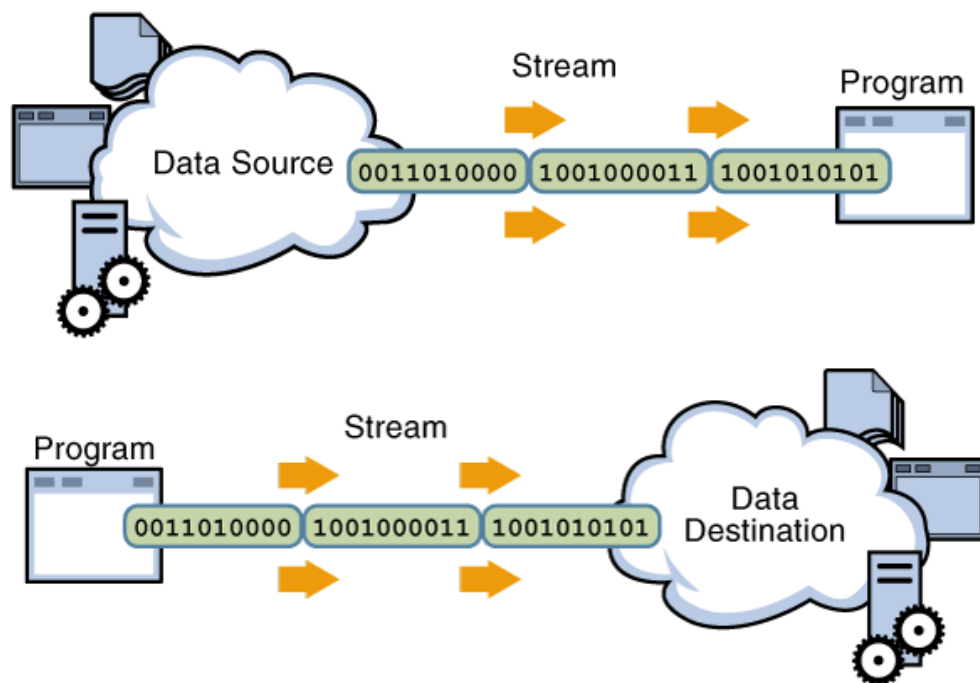


# Anexo UD06: Sockets en la nube (AWS)



## 1. Introducción

## 2. Requisitos

## 3. Guía paso a paso

### 3. 1. Preparar el entorno de la nube

3. 1. 1. Iniciar Laboratorio

3. 1. 2. Crear un entorno Cloud9

3. 1. 3. Creación del servidor de Sockets

3. 1. 4. Abrir el puerto en la instancia EC2 del cloud9

3. 1. 5. Dirección pública de la EC2

### 3. 2. Preparar el cliente local

### 3. 3. Ejecución de prueba

3. 3. 1. Desde el punto de vista del cliente

3. 3. 2. Desde el punto de vista del servidor

## 4. Fuentes de información

# 1. Introducción

---

La intención de este documento es la de dar una perspectiva más realista del uso de sockets, ya que en lugar de usar la misma máquina del alumno como cliente y servidor, vamos a desplegar el servidor del socket en una máquina alojada en la nube de Amazon (AWS).

## 2. Requisitos

---

Para realizar esta práctica guiada necesitamos:

- Acceso al Learner Lab proporcionado por el profesor. (<https://awsacademy.instructure.com>)
- Conocimientos sobre los sockets, IP's y puertos.
- Un dispositivo local con capacidad de ejecutar un cliente de socket, con acceso a los puertos e Ip's de AWS (Ojo con la red de conselleria)

## 3. Guía paso a paso

### 3.1. Preparar el entorno de la nube

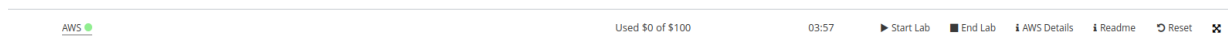
#### 3.1.1. Iniciar Laboratorio

Lo primero que necesitamos es arrancar el laboratorio, para ello Accedemos al LMS del awsacademy, buscamos el Curso facilitado por el docente, accedemos a sus contenidos y a continuación al Learner Lab. (Si es la primera vez que accedemos debemos aceptar los términos de uso).

Inicialmente el laboratorio está en rojo:

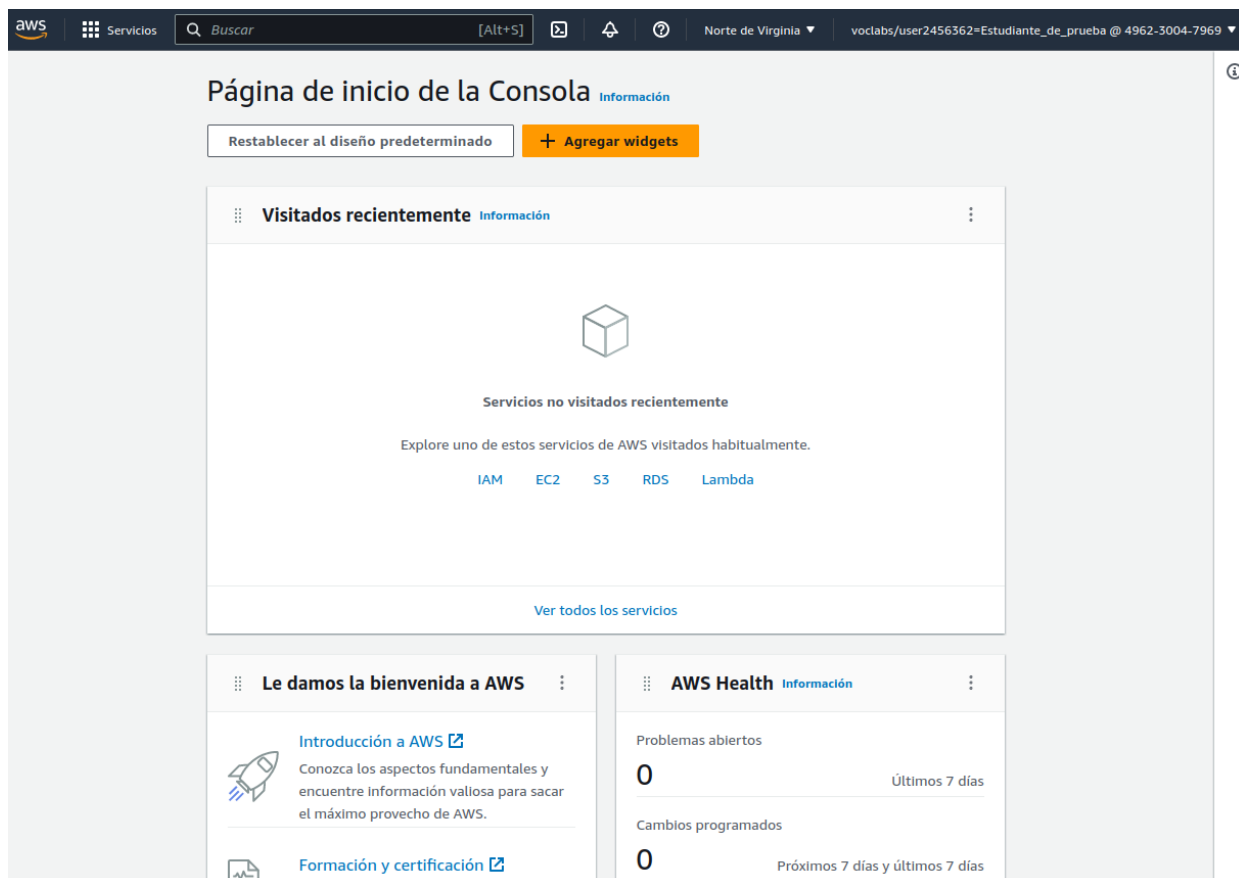


Elegimos la opción **Start Lab** y esperamos a que aparezca el laboratorio en verde:



Por defecto el Learner Lab nos proporciona 100 dolares de saldo, y un tiempo de 4 horas, tras el cual se detendrán la mayoría de servicios que tengamos en marcha. Pero mientras quede saldo podemos volver a iniciar el Laboratorio y dispondremos de 4 horas más.

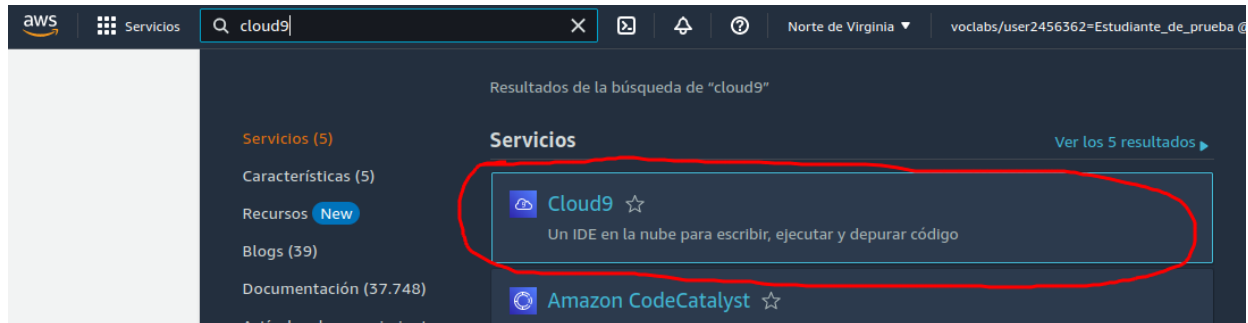
Una vez aparece en verde podemos hacer click sobre las letras AWS y aparecerá el Dashboard de AWS (debemos permitir las ventanas emergentes):



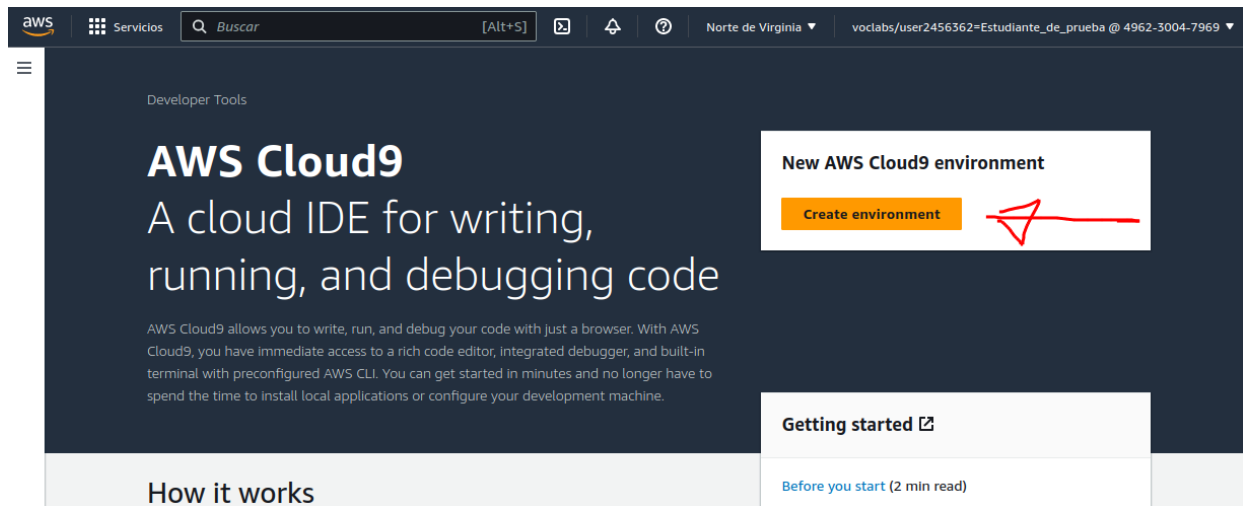
### 3.1.2. Crear un entorno Cloud9

Cloud9 es un entorno de desarrollo en la nube que proporciona AWS asociado a una instancia EC2 (máquina virtual en la nube).

El primer paso es crear este entorno, para ello buscamos cloud9 en la parte superior del Dashboard:



A continuación seleccionamos `Create environment`:



En la siguiente ventana debemos especificar el **nombre** (`Name`), cambiaremos la **plataforma** a `Ubuntu Server 18.04 LTS`, también podemos ampliar el tiempo de Timeout para no tener problemas a `4 horas` i por último dentro de los **Network settings** elegiremos la conexión por `SSH`, el resto de opciones se quedan por defecto y pulsamos el botón naranja del final `Create`.

Create environment [Info](#)

## Details

## Name

cloud9David

Limit of 60 characters, alphanumeric, and unique per user.

Description - *optional*


Limit 200 characters.

Environment type [Info](#)

Determines what the Cloud9 IDE will run on.

☒ New EC2 instance

Cloud9 creates an EC2 instance in your account. The configuration of your EC2 instance cannot be changed by Cloud9 after creation.

☐ Existing compute

You have an existing instance or server that you'd like to use.

## New EC2 instance

Instance type [Info](#)

The memory and CPU of the EC2 instance that will be created for Cloud9 to run on.

☒ t2.micro (1 GIB RAM + 1 vCPU)

Free-tier eligible. Ideal for educational users and exploration.

☐ t3.small (2 GIB RAM + 2 vCPU)

Recommended for small web projects.

☐ m5.large (8 GIB RAM + 2 vCPU)

Recommended for production and most general-purpose development.

☐ Additional instance types

Explore additional instances to fit your need.

Platform [Info](#)

This will be installed on your EC2 instance. We recommend Amazon Linux 2.

Ubuntu Server 18.04 LTS

## Timeout

How long Cloud9 can be inactive (no user input) before auto-hibernating. This helps prevent unnecessary charges.

4 hours

Network settings [Info](#)

## Connection

How your environment is accessed.

☐ AWS Systems Manager (SSM)

Accesses environment via SSM without opening inbound ports (no ingress).

☒ Secure Shell (SSH)

Accesses environment directly via SSH, opens inbound ports.

► VPC settings [Info](#)► Tags - *optional* [Info](#)

A tag is a label that you assign to an AWS resource. Each tag consists of a key and an optional value. You can use tags to search and filter your resources or track your AWS costs.




## The following IAM resources will be created in your account

- **AWSServiceRoleForAWSCloud9** - AWS Cloud9 creates a service-linked role for you. This allows AWS Cloud9 to call other AWS services on your behalf. You can delete the role from the AWS IAM console once you no longer have any AWS Cloud9 environments. [Learn more](#)

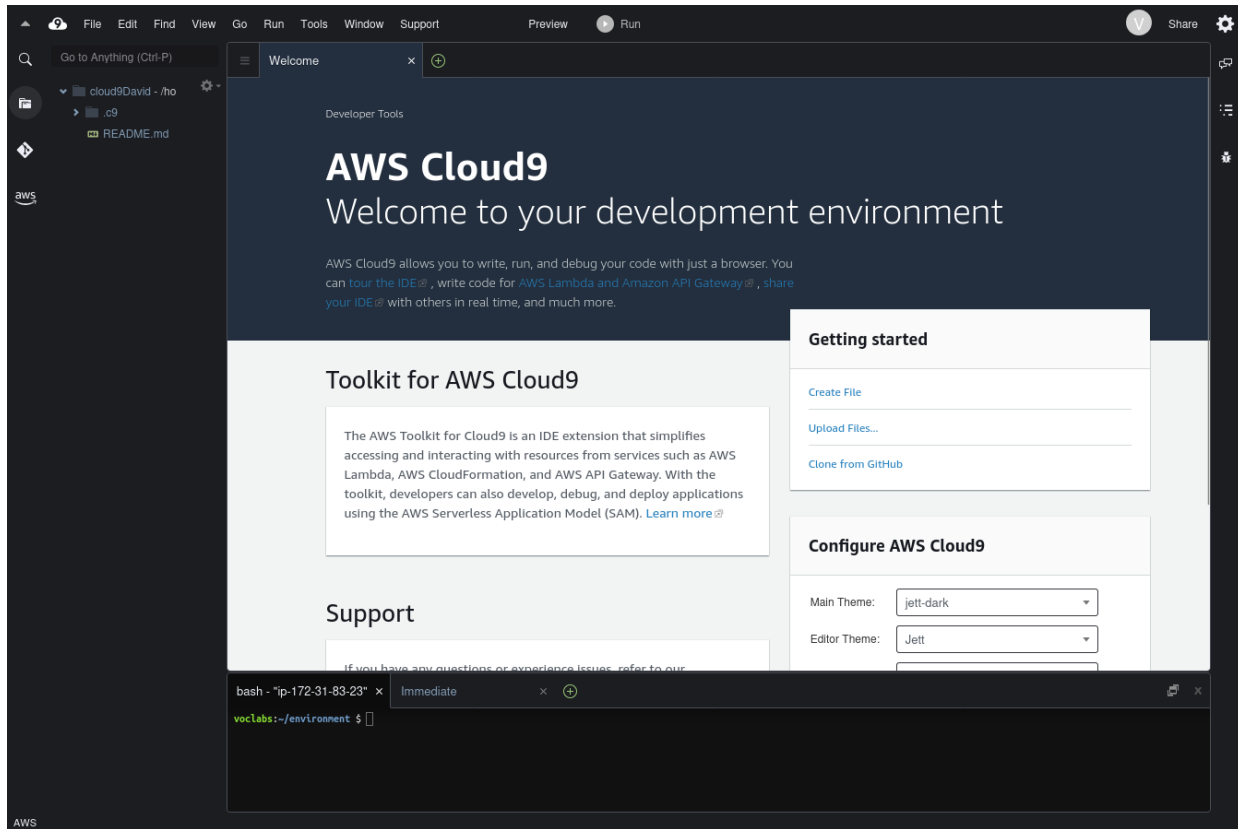
Cancel

Create

Si todo ha ido bien podemos seleccionar el botón **Open**:

Environments (1)						Delete	View details	Open in Cloud9	Create environment
My environments									
Name	Cloud9 IDE	Environment type	Connection	Permission	Owner ARN				
cloud9David	 Open	EC2 Instance	Secure Shell (SSH)	Owner	arn:aws:sts::496230047969:assumed-role/voclabs/user2456362=Estudiante_de_prueba				

Y deberíamos ver algo parecido a esto:



### 3.1.3. Creación del servidor de Sockets

Primero cerraremos la ventana de bienvenida, a continuación creamos un nuevo fichero, por ejemplo `ServidorSocket.java` con el siguiente código java:

```

1  import java.io.*;
2  import java.net.*;
3
4  public class ServidorSocket {
5
6      private static final int PORT=11000;
7
8      public static void main(String[] args) throws IOException, ClassNotFoundException {
9          String FraseClient;
10         String FraseMajusculas;
11         ServerSocket serverSocket;
12         Socket clientSocket;
13         ObjectInputStream entrada;
14         ObjectOutputStream eixida;
15         serverSocket = new ServerSocket(PORT);
16         System.out.println("Server iniciado y escuchando en el puerto "+ PORT);

```



```

17     while (true) {
18         clientSocket = serverSocket.accept();
19         entrada = new ObjectInputStream(clientSocket.getInputStream());
20         FraseClient = (String) entrada.readObject();
21
22         System.out.println("La frase recibida es: " + FraseClient);
23
24         eixida = new ObjectOutputStream(clientSocket.getOutputStream());
25         FraseMajusculas = FraseClient.toUpperCase();
26         System.out.println("El server devuelve la frase: " + FraseMajusculas);
27         eixida.writeObject(FraseMajusculas);
28
29         clientSocket.close();
30         System.out.println("Server esperando una nueva conexión...");
31     }
32 }
33 }

```

Debería quedar algo así:

```

1  import java.io.*;
2  import java.net.*;
3
4  public class ServidorSocket {
5
6      private static final int PORT=11000;
7
8      public static void main(String[] args) throws IOException, ClassNotFoundException {
9          String FraseClient;
10         String FraseMajusculas;
11         ServerSocket serverSocket;
12         Socket clientSocket;
13         ObjectInputStream entrada;
14         ObjectOutputStream eixida;
15         serverSocket = new ServerSocket(PORT);
16         System.out.println("Server iniciado y escuchando en el puerto " + PORT);
17         while (true) {
18             clientSocket = serverSocket.accept();
19             entrada = new ObjectInputStream(clientSocket.getInputStream());
20             FraseClient = (String) entrada.readObject();
21
22             System.out.println("La frase recibida es: " + FraseClient);
23
24             eixida = new ObjectOutputStream(clientSocket.getOutputStream());
25             FraseMajusculas = FraseClient.toUpperCase();
26             System.out.println("El server devuelve la frase: " + FraseMajusculas);
27             eixida.writeObject(FraseMajusculas);
28
29             clientSocket.close();
30             System.out.println("Server esperando una nueva conexión...");
31         }
32     }
33 }

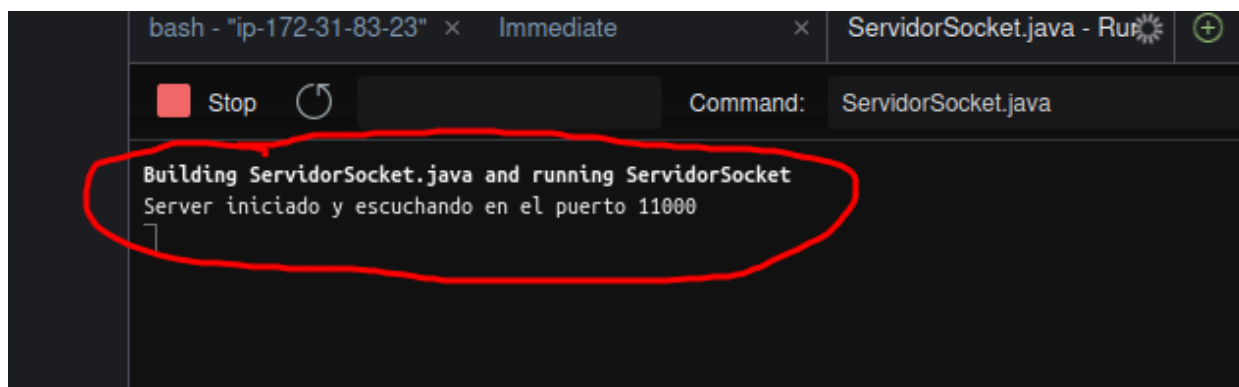
```

bash - "ip-172-31-83-23" x Immediate x

voclabs:~/environment \$

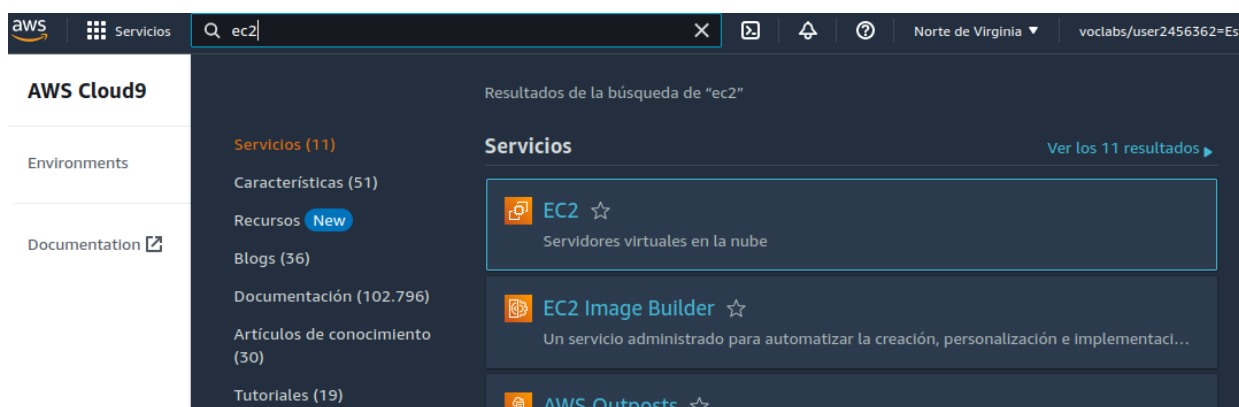
AWS: profile default

Y si iniciamos el servidor:



### 3.1.4. Abrir el puerto en la instancia EC2 del cloud9

Ahora debemos volver a la pestaña donde tenemos el Dashboard de AWS y buscar EC2 (donde antes buscamos cloud9):



Una vez abierto elegimos la opción **Instancias (en ejecución)**:



Deberíamos tener al menos una Instancia, si tenemos más de una debemos buscar la que contenga el nombre de nuestra instancia cloud9, debemos marcar el check que tiene justo delante del nombre y a continuación elegir la pestaña **Seguridad**:

Instancias (1/1) Información

Buscar instancia por atributo o etiqueta (case-sensitive)

Estado de la instancia = running X Quitar los filtros

✓	Name	ID de la instancia	Estado de la i...	Tipo de inst...	Comprobación ...
✓	aws-cloud9-cloud9David-83bd4fb6e5bf401e9b81d1e6f35291c5	i-069949e33eb20baa8	En ejecución	t2.micro	2/2 comprobaciones

Instancia: i-069949e33eb20baa8 (aws-cloud9-cloud9David-83bd4fb6e5bf401e9b81d1e6f35291c5)

Detalles Seguridad Redes Almacenamiento Comprobaciones de estado Monitoreo Etiquetas

▼ Detalles de seguridad

Rol de IAM: - ID del propietario: 496230047969

Grupos de seguridad: sg-0ab93f95e05631a5c (aws-cloud9-cloud9David-83bd4fb6e5bf401e9b81d1e6f35291c5-InstanceSecurityGroup-7JT461Z8I2RR)

▼ Reglas de entrada

Si nos fijamos en las reglas de entrada del grupo de seguridad, solo tiene habilitada la entrada para el puerto 22 (SSH), a continuación hacemos click sobre el enlace del Grupo de seguridad:

#### Grupos de seguridad

[sg-0ab93f95e05631a5c \(aws-cloud9-cloud9David-83bd4fb6e5bf401e9b81d1e6f35291c5-InstanceSecurityGroup-7JT461Z8I2RR\)](#)

#### ▼ Reglas de entrada

Filtrar reglas			
Nombre	ID de la regla del grupo d...	Intervalo de pu...	Protocolo
-	sgr-0cb10956d0b223f9d	22	TCP
-	sgr-09bd2851a6631f7da	22	TCP

Y añadiremos el puerto 11000 (o el que hayamos elegido para nuestro servidor) a las reglas de entrada, elegimos el botón **Editar reglas de entrada**, a continuación **Agregar regla**. Elegimos **TCP Personalizado**, puerto 11000 y **AnywhereIpv4** y añadimos una descripción si lo deseamos:

Editar reglas de entrada Información

Las reglas de entrada controlan el tráfico entrante que puede llegar a la instancia.

Reglas de entrada Información

ID de la regla del grupo de seguridad	Tipo	Protocolo	Intervalo de puertos	Origen	Descripción: opcional	
sgr-0cb10956d0b223f9d	SSH	TCP	22	Personalizada	35.172.155.96/27	Eliminar
sgr-09bd2851a6631f7da	SSH	TCP	22	Personalizada	35.172.155.192/27	Eliminar
-	TCP personalizado	TCP	11000	Anywhere-IPv4	SocketServer	Eliminar

Agregar regla

Cancelar Previsualizar los cambios Guardar reglas

Una vez hecho esto si volvemos a la pestaña Seguridad de nuestra instancia EC2 veremos la regla añadida.

### 3.1.5. Dirección pública de la EC2

Necesitamos saber la DNS de IPv4 pública de nuestra instancia EC2 para acceder desde el cliente, marcamos el check de nuestra instancia y accedemos a la primera pestaña **Detalles**, y nos fijamos en la parte derecha y pulsaremos el botón de copiar y guardaremos esta información para más adelante:

The screenshot shows the AWS Management Console interface for an EC2 instance. The instance is named 'aws-cloud9-cloud9David-83bd4fb6e5bf401e9...' and is in the 'us-east-1' region. The 'DNS de IPv4 pública' is listed as 'ec2-3-84-52-97.compute-1.amazonaws.com'. A red circle highlights this DNS address, and a red arrow points to the 'copiar' button next to it.

## 3.2. Preparar el cliente local

En nuestro IDE preferido creamos un nuevo archivo **ClienteSocket.java** con el siguiente código:

```
1 import java.io.*;
2 import java.net.*;
3 import java.util.Scanner;
4
5 public class ClienteSocket {
6
7     private static final String DNSAWS = "ec2-3-84-52-97.compute-1.amazonaws.com";
8
9     public static void main(String[] args) throws IOException, ClassNotFoundException {
10         Socket socket;
11         ObjectInputStream entrada;
12         ObjectOutputStream eixida;
13         String frase;
14
15         socket = new Socket(DNSAWS, 11000);
16         eixida = new ObjectOutputStream(socket.getOutputStream());
17
18         System.out.println("Introduce la frase a enviar en minúsculas");
19         Scanner in = new Scanner(System.in);
20         frase = in.nextLine();
21         System.out.println("Se envia la frase " + frase);
22         eixida.writeObject(frase);
23
24         entrada = new ObjectInputStream(socket.getInputStream());
```

```

25     System.out.println(
26         "La frase recibida es: " + (String) entrada.readObject());
27     socket.close();
28 }
29 }

```

Recuerda cambiar la constante `DNSAWS` por el `String` que corresponde con la dirección DNS IPv4 de tu instancia EC2 obtenida en el punto 3.1.5.

## 3.3. Ejecución de prueba

### 3.3.1. Desde el punto de vista del cliente

Una vez ejecutado el cliente debe aparecer algo similar a esto:

```

1  Introduce la frase a enviar en minúsculas

```

Escribimos nuestra frase, y al pulsar INTRO obtenemos el siguiente resultado:

```

1  Introduce la frase a enviar en minúsculas
2  esta frase está en minúsculas
3  Se envia la frase esta frase está en minúsculas
4  La frase recibida es: ESTA FRASE ESTÁ EN MINÚSCULAS

```

### 3.3.2. Desde el punto de vista del servidor

La consola de salida del servidor por su parte debe haber registrado la conexión del cliente, la recepción de la frase, y la frase devuelta:

```

1  Server iniciado y escuchando en el puerto 11000
2  La frase recibida es: esta frase está en minúsculas
3  El server devuelve la frase: ESTA FRASE ESTÁ EN MINÚSCULAS
4  Server esperando una nueva conexión...

```

## 4. Fuentes de información

---

- <https://awsacademyinstructure.com>