



**UNIVERSIDAD ARGENTINA DE LA EMPRESA**  
**Departamento de Tecnología Informática**

Ingeniería de Sistemas

Profesor:

Castiñeiras, José Ramón

Trabajo Práctico  
2C-2025

INTEGRANTES

Perrella, Luciano:

1193133

# Consigna 1

Levantar una aplicación **Node.js** en un servidor virtual (AWS EC2) de dos formas:

1. **Manual** → realizar todos los pasos desde la consola AWS y por SSH (instalación de dependencias, configuración de puertos, ejecución del servicio).
2. **Semi-automatizado** → usar el script en **User Data** como primer paso de automatización (script incluido en el apéndice).

## A. Pasos manuales en la consola de AWS (crear y preparar la EC2)

1. Abrir consola AWS → **EC2** → **Instances** → **Launch instances**
2. **AMI**: Ubuntu Server 22.04 LTS (x86\_64) u otra LTS.
3. **Instance type**: por ejemplo **t3.micro** (según requerimientos y cuenta).
4. **Key pair**: seleccionar o crear un par de claves (descargar el **.pem**). Guardarlo en un lugar seguro.
5. **Network settings**:
  - Subnet → elegir una con *Auto-assign Public IPv4* habilitado (o asignar Elastic IP después).
  - Security group (Inbound rules):
    - SSH (TCP 22) — Source: *My IP* (recomendado).
    - HTTP (TCP 80) — Source: **0.0.0.0/0** (si la app debe ser pública).
    - (Opcional) Custom TCP 3001 — Source: *My IP* (solo para acceso directo a Node).
  - Importante: nunca abrir **22** o **3001** a **0.0.0.0/0** en producción sin control.
6. **Storage**: tamaño acorde (ejemplo: 8–20 GB).
7. **Tags**: agregar nombre descriptivo.
8. **Launch**.
9. Esperar a que la instancia esté en *running* y anotar la *Public IPv4* (o asignar un Elastic IP).

## B. Pasos manuales en el servidor (por SSH)

Ejecutar con **sudo** cuando sea requerido.

### 1. Preparar sistema

```
sudo apt update -y
```

```
sudo apt upgrade -y
```

### 2. Instalar utilidades

```
sudo apt install -y git curl build-essential python3 nginx
```

### 3. Instalar Node.js 18

```
curl -fsSL https://deb.nodesource.com/setup_18.x | sudo -E bash -
```

```
sudo apt install -y nodejs
```

### 4. Verificar instalaciones

```
node --version
```

```
npm --version
```

```
git --version
```

```
nginx -v
```

### 5. Clonar repositorio

```
sudo rm -rf /opt/inventory
```

```
sudo git clone https://github.com/LucianoPerrella/inventory /opt/inventory
```

```
sudo chown -R ubuntu:ubuntu /opt/inventory
```

### 6. Instalar dependencias

```
cd /opt/inventory
```

```
sudo -u ubuntu npm install --production
```

## 7. Probar ejecución directa

```
sudo -u ubuntu node server.js &
```

*(detener luego y usar systemd)*

## 8. Crear servicio systemd

Archivo: `/etc/systemd/system/inventory.service`

[Unit]

Description=Inventory App

After=network.target

[Service]

User=ubuntu

WorkingDirectory=/opt/inventory

ExecStart=/usr/bin/node server.js

Restart=always

RestartSec=10

Environment="PORT=3001"

Environment="NODE\_ENV=production"

[Install]

WantedBy=multi-user.target

## 9. Habilitar e iniciar servicio

```
sudo systemctl daemon-reload
```

```
sudo systemctl enable inventory.service
```

```
sudo systemctl start inventory.service
```

## 10. Verificar servicio

```
sudo systemctl status inventory.service --no-pager
```

```
sudo journalctl -u inventory.service -f
```

## 11. Configurar Nginx

Archivo: `/etc/nginx/sites-available/inventory`

```
server {  
  
    listen 80 default_server;  
  
    listen [::]:80 default_server;  
  
    server_name _;  
  
  
    location / {  
  
        proxy_pass http://127.0.0.1:3001;  
  
        proxy_http_version 1.1;  
  
        proxy_set_header Upgrade $http_upgrade;  
  
        proxy_set_header Connection 'upgrade';  
  
        proxy_set_header Host $host;  
  
        proxy_cache_bypass $http_upgrade;  
  
    }  
}
```

```
}
```

## 12. Habilitar sitio y reiniciar Nginx

```
sudo rm -f /etc/nginx/sites-enabled/default
```

```
sudo ln -sf /etc/nginx/sites-available/inventory /etc/nginx/sites-enabled/inventory
```

```
sudo nginx -t
```

```
sudo systemctl enable nginx
```

```
sudo systemctl restart nginx
```

## 13. Verificar puertos y conectividad

```
ss -tuln | grep 3001 || echo "Puerto 3001 no detectado"
```

```
curl -I http://127.0.0.1:3001
```

```
curl -I http://127.0.0.1
```

## 14. Probar desde máquina local

```
curl -I http://<PUBLIC_IP>
```

# Consigna 1 – Instancia automatizada

**Instancia:** [ec2-3-84-200-127.compute-1.amazonaws.com](https://ec2-3-84-200-127.compute-1.amazonaws.com)

Se debe modificar **User Data** para que ejecute el siguiente script de instalación automática (Node.js + servicio + Nginx).

```
#!/bin/bash
```

```
set -e
```

```
# Crear log desde el principio
```

```
touch /var/log/user-data.log
```

```
exec > /var/log/user-data.log 2>&1
```

```
echo "=====
```

```
echo "INICIO INSTALACION: $(date)"
```

```
echo "=====
```

```
export DEBIAN_FRONTEND=noninteractive
```

```
# Actualizar sistema e instalar dependencias
```

```
echo "Actualizando sistema..."
```

```
apt-get update -y
```

```
apt-get install -y git curl build-essential python3 nginx
```

```
# Instalar Node.js
```

```
echo "Instalando Node.js..."
```

```
curl -fsSL https://deb.nodesource.com/setup_18.x | bash -
```

```
apt-get install -y nodejs
```

```
echo "Node version: $(node --version)"
```

```
echo "NPM version: $(npm --version)"
```

```
# Clonar repositorio
```

```
echo "Clonando repositorio..."
```

```
rm -rf /opt/inventory
```

```
git clone https://github.com/LucianoPerrella/inventory /opt/inventory
```

```
# Configurar permisos
```

```
echo "Configurando permisos..."
```

```
chown -R ubuntu:ubuntu /opt/inventory
```

```
# Instalar dependencias de producción
```

```
echo "Instalando dependencias npm..."
```

```
cd /opt/inventory
```

```
sudo -u ubuntu npm install --production
```

```
# Verificar server.js
```

```
echo "Verificando server.js..."
```

```
ls -la server.js
```



```
# Crear servicio systemd
```

```
echo "Creando servicio systemd..."
```

```
cat > /etc/systemd/system/inventory.service <<'EOF'
```

```
[Unit]
```

```
Description=Inventory App
```

```
After=network.target
```

```
[Service]
```

```
User=ubuntu
```

```
WorkingDirectory=/opt/inventory
```

```
ExecStart=/usr/bin/node server.js
```

```
Restart=always
```

```
RestartSec=10
```

```
Environment="PORT=3001"
```

```
Environment="NODE_ENV=production"
```

```
Environment=DB_HOST=inventory.cz6imua8gy2g.us-east-1.rds.amazonaws.com
```

```
Environment=DB_PORT=5432
```

```
Environment=DB_USER=postgres
```

```
Environment=DB_PASS=inventory
```

```
Environment=DB_NAME=inventory
```

```
[Install]
```

```
WantedBy=multi-user.target
```

EOF

```
echo "Contenido del archivo de servicio:"
```

```
cat /etc/systemd/system/inventory.service
```

```
# Iniciar servicio
```

```
echo "Iniciando servicio..."
```

```
systemctl daemon-reload
```

```
systemctl enable inventory.service
```

```
systemctl start inventory.service
```

```
# Esperar inicialización
```

```
echo "Esperando que inicie..."
```

```
sleep 15
```

```
echo "Estado del servicio:"
```

```
systemctl status inventory.service --no-pager || true
```

```
echo "Logs del servicio:"
```

```
journalctl -u inventory.service -n 20 --no-pager || true
```

```
# Configurar Nginx
```

```
echo "Configurando Nginx..."
```

```
cat > /etc/nginx/sites-available/inventory <<'EOF'
```

```
server {  
  
    listen 80 default_server;  
  
    listen [::]:80 default_server;  
  
    server_name _;  
  
  
    location / {  
  
        proxy_pass http://127.0.0.1:3001;  
  
        proxy_http_version 1.1;  
  
        proxy_set_header Upgrade $http_upgrade;  
  
        proxy_set_header Connection 'upgrade';  
  
        proxy_set_header Host $host;  
  
        proxy_cache_bypass $http_upgrade;  
  
    }  
}
```

EOF

# Habilitar sitio

```
echo "Habilitando sitio..."
```

```
rm -f /etc/nginx/sites-enabled/default
```

```
ln -sf /etc/nginx/sites-available/inventory /etc/nginx/sites-enabled/inventory
```

# Testear y reiniciar Nginx

```
echo "Testeando configuración Nginx..."
```

```
nginx -t
```

```
echo "Reiniciando Nginx..."
```

```
systemctl enable nginx
```

```
systemctl restart nginx
```

```
# Obtener IP pública
```

```
PUBLIC_IP=$(curl -s http://169.254.169.254/latest/meta-data/public-ipv4 || echo  
"NO_DISPONIBLE")
```

```
echo "=====
```

```
echo "INSTALACION COMPLETADA: $(date)"
```

```
echo "Aplicacion disponible en: http://$PUBLIC_IP"
```

```
echo "=====
```

#### Aclaraciones

- Este script es una **modificación del User Data existente**, que actualmente integra Postgres y RDS.
- La **corrección principal** realizada fue:
  - Quitar las comillas dobles que encerraban el valor de la URL de la base de datos (**DB\_HOST**) para que Node.js pueda conectarse correctamente.
- La aplicación se ejecuta en **puerto 3001**, mientras que Nginx la expone en el **puerto 80**.
- La **URL de acceso** será: **http://<PUBLIC\_IP>** (la IP pública que devuelve AWS).

# Consigna 2

- Crear una instancia en Amazon EC2 usando la CLI de AWS desde la laptop, sin utilizar la consola web.
- Instalar y desplegar la Inventory App en la instancia.
- Documentar los comandos utilizados y las configuraciones necesarias.

## Resumen de la solución

Se utilizó un script Bash que automatiza todo el flujo:

1. Obtiene la AMI de Ubuntu 22.04 más reciente para la región especificada.
2. Crea (o reemplaza) un Key Pair y lo guarda localmente (.pem).
3. Crea un Security Group y configura reglas de firewall (SSH, HTTP y puerto de la app).
4. Genera un userdata.sh con pasos para instalar Node.js, clonar el repositorio y configurar Nginx + systemd.
5. Lanza la instancia EC2 con --user-data file:///userdata.sh.
6. Espera a que la instancia esté en estado running y devuelve la IP pública y datos de acceso.

## Supuestos:

- AWS CLI configurada con perfil y credenciales.
- Permisos para crear recursos (EC2, SG, KeyPair).
- Acceso a Internet desde la laptop.
- Se utiliza usuario ubuntu según la AMI seleccionada.

```
#!/bin/bash
set -e
```

```
echo "=====
echo "DESPLIEGUE DE INVENTORY APP VIA AWS CLI"
echo "=====
echo ""
```

```
# =====
# CONFIGURACION
```

```

# =====
REGION="us-east-1"
SG_NAME="inventory-sg-cli"
KEY_NAME="inventory-key-cli"
INSTANCE_NAME="inventory-app-cli"
INSTANCE_TYPE="t2.micro"
APP_PORT=3001

echo "Configuración:"
echo "  Región: $REGION"
echo "  Security Group: $SG_NAME"
echo "  Key Pair: $KEY_NAME"
echo "  Tipo instancia: $INSTANCE_TYPE"
echo ""

# =====
# 1. OBTENER AMI DE UBUNTU 22.04
# =====
echo "[1/7] Obteniendo AMI de Ubuntu 22.04..."
AMI_ID=$(aws ec2 describe-images \
  --region $REGION \
  --owners 099720109477 \
  --filters "Name=name,Values=ubuntu/images/hvm-ssd/ubuntu-jammy-22.04-amd64-server-*" \
    "Name=state,Values=available" \
  --query 'Images | sort_by(@, &CreationDate) | [-1].ImageId' \
  --output text)

if [ -z "$AMI_ID" ]; then
  echo "ERROR: No se pudo obtener AMI"
  exit 1
fi
echo "  AMI ID: $AMI_ID"
echo ""

# =====
# 2. CREAR KEY PAIR
# =====
echo "[2/7] Creando Key Pair..."
if aws ec2 describe-key-pairs --region $REGION --key-names $KEY_NAME &>/dev/null; then
  echo "  Key pair ya existe, eliminando..."
  aws ec2 delete-key-pair --region $REGION --key-name $KEY_NAME
fi

aws ec2 create-key-pair \

```

```
--region $REGION \  
--key-name $KEY_NAME \  
--query 'KeyMaterial' \  
--output text > $KEY_NAME.pem
```

```
chmod 400 $KEY_NAME.pem  
echo " Key guardada en: $KEY_NAME.pem"  
echo ""
```

```
# =====  
# 3. CREAMAR SECURITY GROUP  
# =====
```

```
echo "[3/7] Creando Security Group..."  
SG_EXISTING=$(aws ec2 describe-security-groups \  
  --region $REGION \  
  --filters "Name=group-name,Values=$SG_NAME" \  
  --query 'SecurityGroups[0].GroupId' \  
  --output text 2>/dev/null || echo "None")
```

```
if [ "$SG_EXISTING" != "None" ] && [ -n "$SG_EXISTING" ]; then  
  echo " Security Group existente, eliminando..."  
  aws ec2 delete-security-group --region $REGION --group-id $SG_EXISTING  
fi
```

```
SG_ID=$(aws ec2 create-security-group \  
  --region $REGION \  
  --group-name $SG_NAME \  
  --description "Security group para Inventory App" \  
  --query 'GroupId' \  
  --output text)
```

```
echo " Security Group ID: $SG_ID"
```

```
# =====  
# 4. CONFIGURAR REGLAS DE FIREWALL  
# =====
```

```
echo "[4/7] Configurando reglas de firewall..."  
aws ec2 authorize-security-group-ingress --region $REGION --group-id $SG_ID --protocol tcp  
--port 22 --cidr 0.0.0.0/0  
echo " ✓ Puerto 22 (SSH) abierto"  
aws ec2 authorize-security-group-ingress --region $REGION --group-id $SG_ID --protocol tcp  
--port 80 --cidr 0.0.0.0/0  
echo " ✓ Puerto 80 (HTTP) abierto"
```

```
aws ec2 authorize-security-group-ingress --region $REGION --group-id $SG_ID --protocol tcp
--port $APP_PORT --cidr 0.0.0.0/0
echo "  ✓ Puerto $APP_PORT (App) abierto"
echo ""
```

```
# =====
```

```
# 5. CREAR SCRIPT USER DATA
```

```
# =====
```

```
echo "[5/7] Generando script de User Data..."
```

```
cat > userdata.sh <<'USERDATA'
```

```
#!/bin/bash
```

```
set -e
```

```
# ... (Aquí iría exactamente el contenido del User Data que instala Node, clona el repo,
configura systemd y Nginx)
```

```
USERDATA
```

```
echo " User Data guardado en: userdata.sh"
```

```
echo ""
```

```
# =====
```

```
# 6. LANZAR INSTANCIA EC2
```

```
# =====
```

```
echo "[6/7] Lanzando instancia EC2..."
```

```
INSTANCE_ID=$(aws ec2 run-instances \
```

```
--region $REGION \
```

```
--image-id $AMI_ID \
```

```
--instance-type $INSTANCE_TYPE \
```

```
--key-name $KEY_NAME \
```

```
--security-group-ids $SG_ID \
```

```
--user-data file://userdata.sh \
```

```
--tag-specifications "ResourceType=instance,Tags=[{Key=Name,Value=$INSTANCE_NAME}]"
```

```
\
```

```
--query 'Instances[0].InstanceId' \
```

```
--output text)
```

```
echo " Instance ID: $INSTANCE_ID"
```

```
echo ""
```

```
# =====
```

```
# 7. ESPERAR Y OBTENER IP PUBLICA
```

```
# =====
```

```
echo "[7/7] Esperando que la instancia inicie..."
```

```
aws ec2 wait instance-running --region $REGION --instance-ids $INSTANCE_ID
```



```
PUBLIC_IP=$(aws ec2 describe-instances --region $REGION --instance-ids $INSTANCE_ID  
--query 'Reservations[0].Instances[0].PublicIpAddress' --output text)
```

```
echo ""  
echo "INSTALACION COMPLETADA"  
echo " Instance ID: $INSTANCE_ID"  
echo " IP Pública: $PUBLIC_IP"  
echo " Via Nginx (puerto 80): http://$PUBLIC_IP"  
echo " Directo (puerto 3001): http://$PUBLIC_IP:3001"  
echo " SSH: ssh -i $KEY_NAME.pem ubuntu@$PUBLIC_IP"
```

# Consigna 3

- Preparar la aplicación para desplegarla en **Elastic Beanstalk (EB)**.
- Ejecutar el despliegue y comprobar que la app queda disponible en la **URL asignada**.
- Destacar las **diferencias** frente a desplegarla en **EC2** (manual/CLI).

## Elastic Beanstalk (EB) – Conceptos

Elastic Beanstalk es una **plataforma PaaS** que gestiona automáticamente:

- Provisioning de infraestructura.
- Despliegue de la aplicación.
- Balanceo de carga.
- Auto scaling.
- Health checks y rollbacks automáticos.

Para una app **Node.js**, normalmente **no necesitas Docker**, ya que EB usa el script **start** de **package.json** o un **Procfile** para arrancar.

### Flujo típico de despliegue:

1. Preparar el proyecto para EB (archivos mínimos).
2. Instalar y configurar **EB CLI** (o usar la consola/CLI de AWS).
3. Comandos principales: **eb init** → **eb create** → **eb open** (o usar **aws elasticbeanstalk** para versiones/entorno).
4. Verificar URL y logs.

- 
1. Preparar el proyecto local

Antes de subirlo a EB:

- Verificar que `package.json` tenga un script `start` válido:

```
"scripts": {  
  "start": "node server.js"  
}
```

- Comprimir los archivos del proyecto en un `.zip`, **sin incluir** `node_modules/` ni `.git/`.

Ejemplo: seleccionar `server.js`, `package.json`, `Procfile` (si existe) → clic derecho → “Comprimir en `.zip`”.

## 2. Entrar a Elastic Beanstalk

- Ingresar a **AWS Console** → buscar **Elastic Beanstalk**.
- Hacer clic en **Create application**.

## 3. Crear la aplicación

- **Application name:** `inventory-app`
- **Descripción (opcional):** breve texto.
- **Platform:** Node.js
- **Platform branch:** Node.js 18 running on 64bit Amazon Linux 2 (o la más reciente).
- **Application code:** seleccionar **Upload your code** y subir el `.zip` del proyecto.

## 4. Configurar el entorno

- **Environment name:** `inventory-env`

- **Domain:** personalizar subdominio si se desea, por ejemplo: `inventory-env`
- URL resultante:

`http://inventory-env.us-east-1.elasticbeanstalk.com`

---

## 5. Lanzar el entorno

- Hacer clic en **Create environment**.
- Elastic Beanstalk realizará automáticamente:
  - Crear un **Security Group**.
  - Crear un **balanceador de carga** (si no es single instance).
  - Crear un **EC2** con tu aplicación.
  - Configurar **health checks**.

Este proceso puede tardar entre 5 y 10 minutos.

---

## 6. Verificar la aplicación

- Una vez que el estado aparezca como **Health: Ok**, hacer clic en la **URL del entorno**:

`http://inventory-env.us-east-1.elasticbeanstalk.com`

- La aplicación debería estar corriendo correctamente.
- 

Diferencias clave: Elastic Beanstalk vs EC2 (manual/CLI)

## Elastic Beanstalk (PaaS – administrado)

- Abstracción/Automatización: EB gestiona provisioning, balanceo, auto-scaling, health checks, rollbacks automáticos y versioning.
- Despliegue simplificado: `eb deploy` o subir un ZIP, EB instala dependencias y ejecuta `start`.
- Menos configuración manual: no necesitás crear systemd ni configurar Nginx manualmente.
- Escalabilidad automática: soporte out-of-the-box para Auto Scaling y Load Balancer.
- Menos control de bajo nivel: aunque podés personalizar con `.ebextensions` o `.platform`.
- Ideal para: despliegues rápidos, staging, aplicaciones donde no querés manejar infraestructura diaria.

## EC2 (IaaS – manual/CLI)

- Control total: administrás AMI, paquetes, systemd, Nginx, configuraciones de OS y seguridad.
- Mayor complejidad: balanceo, scaling, monitoreo y actualización se gestionan manualmente o con scripts.
- Mayor responsabilidad: backups, parches, disponibilidad, logging y escalado dependen de vos.
- Mejor cuando: necesitás personalizaciones profundas del OS, dependencias nativas específicas, o arquitecturas fuera del patrón web app estándar.

# Consigna 4

## Inventario de Productos (Node.js + Express + PostgreSQL/RDS)

### Descripción General

Inventory App es una **aplicación de gestión de inventario** basada en **Node.js** y **Express**. Su función principal es permitir operaciones **CRUD** (Crear, Leer, Actualizar, Eliminar) sobre productos, almacenados de manera persistente en una base de datos **PostgreSQL**.

**Objetivo clave:** demostrar la **automatización del despliegue** en un servidor virtual, utilizando **Amazon Web Services (AWS)**.

### Stack Tecnológico

La aplicación se basa en los siguientes componentes:

- **Backend:** Node.js (v18), entorno de ejecución del servidor.
- **Framework:** Express.js, para crear la API REST.
- **Base de Datos:** PostgreSQL (RDS), almacenamiento persistente y escalable.
- **Driver DB:** **pg** (node-postgres), cliente para interactuar con la base de datos.
- **Proxy Web:** Nginx, configurado como **proxy inverso**, redirigiendo tráfico HTTP (puerto 80) al puerto 3001 de la aplicación Node.js.

### Despliegue Automatizado en AWS

El despliegue en la **instancia EC2** se realiza mediante el **script User Data** de AWS, permitiendo que la aplicación se aprovisiona automáticamente.

#### 1. Pre-requisitos de Infraestructura y Red

Para un despliegue exitoso, se requiere:

- **Instancia EC2 (Ubuntu):** Servidor principal de la aplicación.
- **Instancia RDS (PostgreSQL):** Base de datos persistente.

- **Configuración de Grupos de Seguridad:**
  - **RDS Inbound:** permitir tráfico entrante en el puerto 5432 solo desde el **Security Group** del EC2.
  - **EC2 Outbound:** permitir tráfico saliente en el puerto 5432 hacia RDS.
  - **EC2 Inbound:** permitir tráfico entrante en el puerto 80 (HTTP) desde Internet.

## 2. Archivos Clave del Despliegue

Componentes esenciales para el funcionamiento:

- **server.js:** Contiene la lógica de conexión a PostgreSQL, manejo de errores, y función de **bootstrapping**, creando la tabla **products** e insertando datos de ejemplo si la DB está vacía.
- **user\_data.sh:** Script de automatización que aprovisiona el servidor (instala Node.js, configura Systemd y Nginx).
- **inventory.service:** Archivo de **Systemd** que gestiona la aplicación como un servicio en segundo plano, asegurando reinicio automático ante fallos.

## 3. Fases del Proceso Automatizado (User Data)

El **script User Data** realiza las siguientes tareas:

1. **Instalación:** Node.js, Nginx, Git y utilidades necesarias.
2. **Clonación y Dependencias:** Clona el repositorio e instala dependencias NPM.
3. **Configuración de Systemd:** Crea **inventory.service** e inyecta variables de entorno de RDS (**DB\_HOST**, **DB\_USER**, etc.) para habilitar la conexión.
4. **Inicio del Servicio:** Habilita e inicia **inventory.service**.
5. **Proxy Nginx:** Configura Nginx para redirigir tráfico del puerto 80 al puerto 3001 donde corre la aplicación.

## Notas Adicionales

- **Persistencia:** Garantizada mediante PostgreSQL.
- **Bootstrapping:** `server.js` autoinicializa el esquema de la base de datos, creando la tabla si no existe, facilitando despliegues en entornos nuevos.
- **Monitoreo:** Verificar el estado del servicio en EC2 con:

```
sudo systemctl status inventory.service
```



# Evidencia de instancias creadas

Instancias (4) Información

Buscar instancia por atributo o etiqueta (case-sensitive)

Todos los ...

Conectar

Estado de la instancia

Acciones

Lanzar instancias

<input type="checkbox"/>	Name	ID de la instancia	Estado de la ...	Tipo de inst...	Comprobación de	Estado de la al:	Zona de dispon...	DNS de IPv4 pública	Di
<input type="checkbox"/>	app-manual-deployment	i-0ff6cee0a97741f8e	En ejecución	t2.micro	2/2 comprobador	Ver alarmas +	us-east-1b	ec2-3-91-7-91.compute...	3.
<input type="checkbox"/>	inventory-app-cli	i-0e3e9a68eca7fb24b	En ejecución	t2.micro	2/2 comprobador	Ver alarmas +	us-east-1b	ec2-98-88-25-22.comp...	98
<input type="checkbox"/>	Inventory-elasticbean-env	i-09b7c0f36afa24e2d	En ejecución	t3.micro	3/3 comprobador	Ver alarmas +	us-east-1b	ec2-52-23-135-235.co...	52