ACCESO A DATOS

SVA. Dev containers

Práctica 7 (no evaluable)

Contenido

Práctica voluntaria

Objetivos:

- Familiarizarse con el uso de “dev containers” para el desarrollo de aplicaciones desplegados en servidores remotos (nube aws).

Materiales y recursos:

- Interprete de comandos: MobaXterm.
- Cuenta de estudiante de awsacademy y acceso al learner lab.
- Software: IDE Visual Studio Code: https://code.visualstudio.com/
- Extensión *Dev Containers* en Visual Studio Code.
- Extensión *Remote Explorer* en Visual Studio Code.

Introducción a los dev containers:

Los “**dev containers**” (development containers) son entornos ligeros y aislados que utilizan la tecnología de Docker para empaquetar un entorno de desarrollo completo dentro de un contenedor. Contienen todo el software necesario preinstalado en un entorno virtual, listo para usar (JDK, dependencias, configuración, etc).

Los desarrollos profesionales son siempre colaborativos (participan numerosos desabolladores). Los *dev containers* ayudan a que todos en un equipo utilicen la misma configuración para su entorno de desarrollo, sin importar qué computadora estén usando.

La imagen base del contenedor puede ser personalizada en función del lenguaje de programación, el tipo de aplicación a desarrollar, las extensiones de VSCode que queramos tener disponibles de partida, ...

Los archivos de configuración de Dev Container (devcontainer.json) se encuentran en la carpeta .devcontainer en la raíz de tu proyecto de forma que

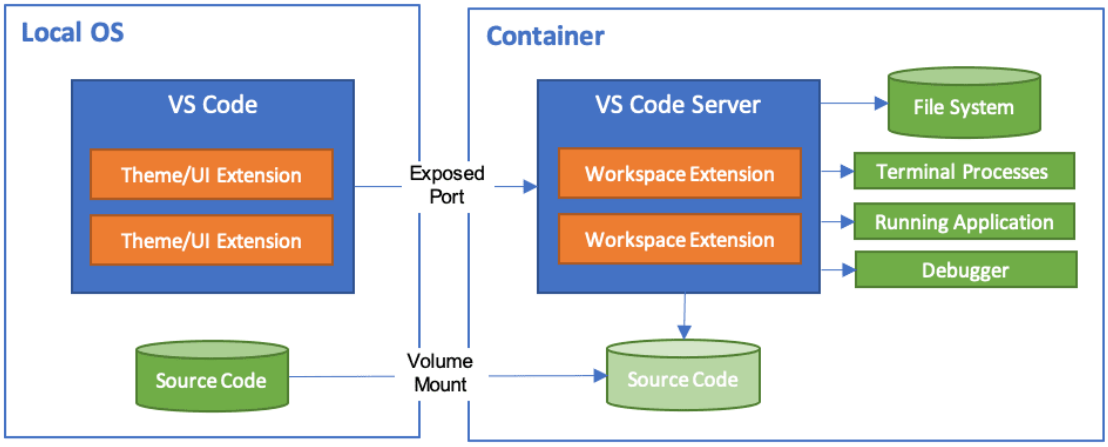
nuestro VSCode local lo detecte y nos ofrezca la opción de **"Reabrir en contenedor"** el proyecto.

En los entornos de desarrollo locales tradicionales, se instala todo manualmente y a menudo se enfrentan a desafíos relacionados con problemas de compatibilidad, conflictos de dependencias y versiones, y la necesidad de una configuración y configuración extensiva.

Ideas clave:

- El **código** debe estar en la máquina remota donde reside Docker por lo que será allí, mediante remote explorer (ssh), donde clonaremos el repositorio sobre el que trabajar.
- El VSCode se divide en dos:
    - **VSCode local** no necesita JDKs, ni Maven, ni ninguna extensión salvo "dev containers" y "remote explorer". Solo hace labor de interfaz.
    - **VSCode servidor** es el que "hace todo" y está instalado automáticamente dentro del contenedor de la máquina remota.
- El contenedor accede al código mediante un volumen (**Bind Mounts)** que básicamente, "mapea" una carpeta del disco duro remoto a una ruta dentro del contenedor (generalmente, /workspaces/mi-repositorio

Arquitectura básica (asumiendo Docker desplegado localmente)



Desarrollo de la actividad:

A continuación, se detallan los pasos necesarios para instalar Docker en la nube (solo debemos hacerlo si no hemos hecho la práctica previa (AADD_SVAP06_DOCKER_JDBC) y posteriormente desplegar un dev container para trabajar sobre un proyecto existente.

Fase 1 (instalación Docker en AWS)

1. Crear una instancia EC2 que llames aadd 2526 docker: selecciona como **SO Ubuntu** y una arquitectura **t3.small**. Genera un **par de claves RSA ppk** llamadas AADD2526 y descárgalas a local. Dichas claves nos permitirán conectarnos a la instancia EC2 mediante una terminal (MobaXterm) con el usuario *ubuntu* sin necesidad de utilizar contraseña. En el grupo de seguridad de la instancia EC2 habilita el **puerto 22** (ssh). Fija un tamaño de disco de **16Gb** y asigna a la instancia una **ip elástica** para que no varíe si accedemos repetidas veces a la máquina en distintas sesiones.

2. Una vez desplegada de la instancia EC2, debes acceder a ella desde **MobaTerm** (https://mobaxterm.mobatek.net/download-home-edition.html). Para ello, crea una nueva sesión SSH fijando los siguientes valores:

   - **Remote host**: debes indicar la ip pública de la instancia EC2
   - **Specify username**: Ubuntu
   - En **advanced SSH settings**, activa la opción "**use private key**" y localiza el **fichero ppk** descargado durante la creación de la instancia EC2.

   Nota: Puedes descargar la versión portable para no tener que instalarla en el ordenador de clase.

3. Una vez accedas a la instancia, instala Docker (gestor de contenedores) siguiendo los siguientes pasos:

   sudo apt update

   sudo apt install ca-certificates curl gnupg

   sudo install -m 0755 -d /etc/apt/keyrings

   *curl -fsSL https://download.docker.com/linux/ubuntu/gpg | *
   sudo gpg --dearmor -o /etc/apt/keyrings/docker.gpg

   sudo chmod a+r /etc/apt/keyrings/docker.gpg

```
echo \
"deb [arch=$(dpkg --print-architecture) \
signed-by=/etc/apt/keyrings/docker.gpg] \
https://download.docker.com/linux/ubuntu \
$(. /etc/os-release && echo $VERSION_CODENAME) stable" | \
sudo tee /etc/apt/sources.list.d/docker.list > /dev/null

sudo apt update

sudo apt install -y docker-ce docker-ce-cli containerd.io docker-buildx-plugin docker-compose-plugin
```

4. Instalado el software, ahora añade el usuario Ubuntu al grupo *docker* (para no tener que hacer sudo cada vez). Asegúrate de NO haber hecho un *sudo su* porque esto siguiente has de ejecutarlo como usuario ubuntu. Si has hecho *sudo su*, debes ejecutar exit para volver a ser usuario Ubuntu.

```
sudo usermod -aG docker $USER

newgrp docker
```

Verifica que tu usuario (Ubuntu) ahora pertenece también al grupo docker con el comando **id.** Deberías ver algo como:

uid=1000(ubuntu) gid=988(docker) groups=**988(docker)**,4(adm),24(cdrom),27(sudo),30(dip),105(lxd),1000(ubuntu)

5. Actualizado el usuario ubuntu, prueba a ejecutar tu primer contenedor de docker. Para ello usaremos una imagen muy básica disponible en el repositorio general de Docker: https://hub.docker.com/_/hello-world. (Necesitas conexión a internet)

   La sentencia para arrancar el contenedor es:

   docker run hello-world

   Se arrancará un nuevo contenedor que mostrará el mensaje: *Hello from Docker!*

   Si consigues visualizar le mensaje, sabrás que el gestor de contenedores Docker está funcionando correctamente.

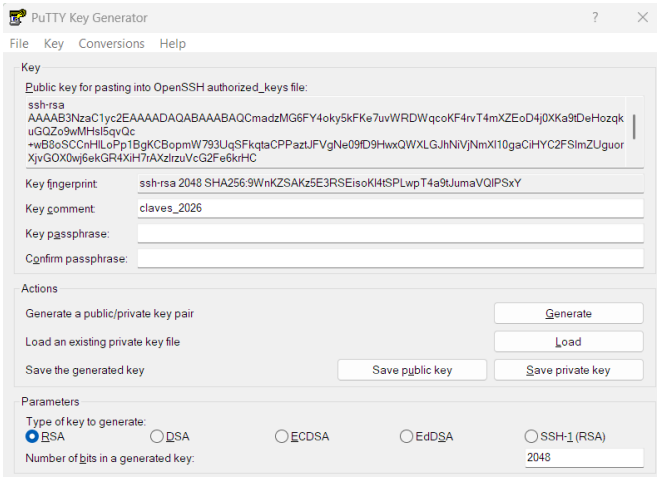
Fase 2 (configuración VSCode + despliegue dev container)

1. Localmente usaremos un VSCode “light”, es decir, sin JDK, ni Maven, ni ninguna extensión salvo las indispensables para trabajar con dev-containers (SSH remote, dev contaniner y Java pack (este último no tiene JDK). En el AV tenéis disponible el entorno portable que usaremos:

   https://cloud.educa.madrid.org/s/8WKC2seYKjqRAYr

2. Descomprimimos el entorno de desarrollo, pero **NO LO EJECUTAMOS AÚN.**

3. Para conectarnos a la instancia EC2 de AWS necesitamos la clave privada en formato OpenSSH. Los pasos para poder conectarnos mediante la extensión “ssh remote” de VSCode son:
    a. Descargar putty-gen (si no lo tienes ya):
        https://the.earth.li/~sgtatham/putty/latest/w64/puttygen.exe
    b. Desde el putty-gen, importar el fichero ppk de claves generado en AWS para el acceso a la instancia EC2. Menú “conversions” -> “import key”

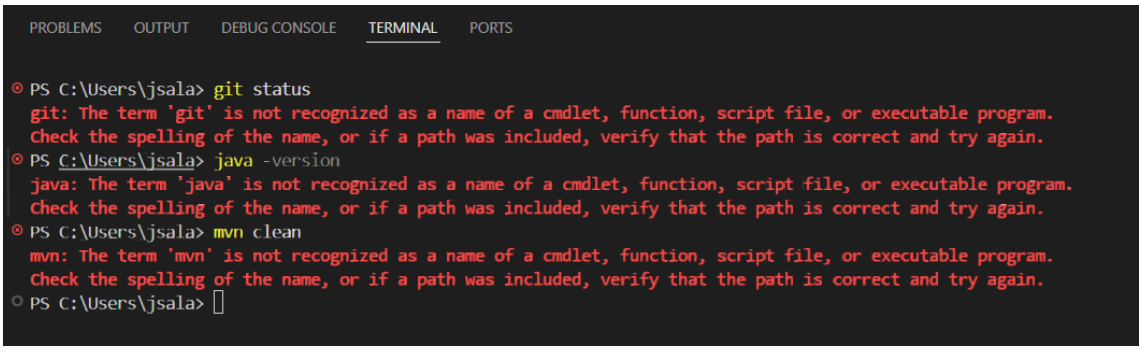


    c. Exportamos en formato OpenSSH (sin contraseña): Menú “conversions” -> “Export OpenSSH Key” y ubicamos el nuevo fichero en la ruta: **C:\Users\XXX\.ssh** poniendole nombre **id_aws_docke**. Siendo XXX tu cuenta de usuario Windows.
    d. En la ruta **C:\Users\XXX\.ssh** crearemos un fichero (si no existe) llamado **config** y añadiremos el siguiente contenido;

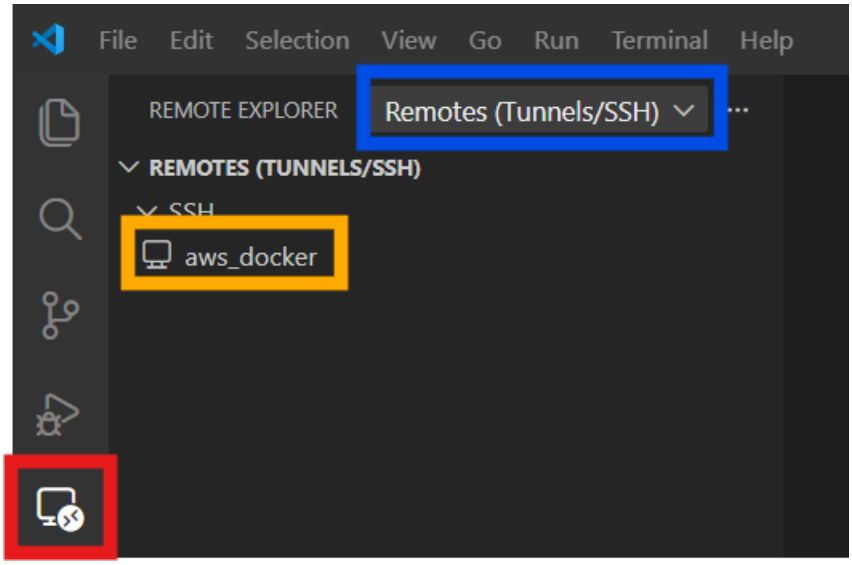
```
Host aws_docker
        HostName 34.224.249.198
        User ubuntu
        IdentityFile ~/.ssh/id_aws_docker
        IdentitiesOnly yes
        StrictHostKeyChecking accept-new
```

Nota: *Hostname* debe ser la ip pública de la instancia EC2 donde has instalado Docker. *User* es Ubuntu porque en nuestra EC2 hemos instalado Ubuntu, si fuera otra distro de Linux, habría que fijarle el usuario correspondiente.

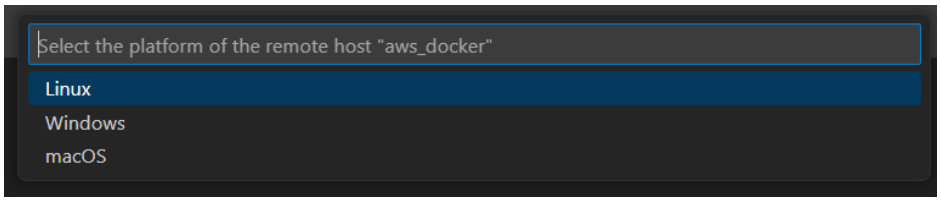
4. Ahora ya sí arrancamos el entorno portable mediante doble clic en “launch.cmd”, y verificamos que no tenemos disponibles las herramientas básicas de nuestros desarrollos de Java en nuestro VSCode “light” local:



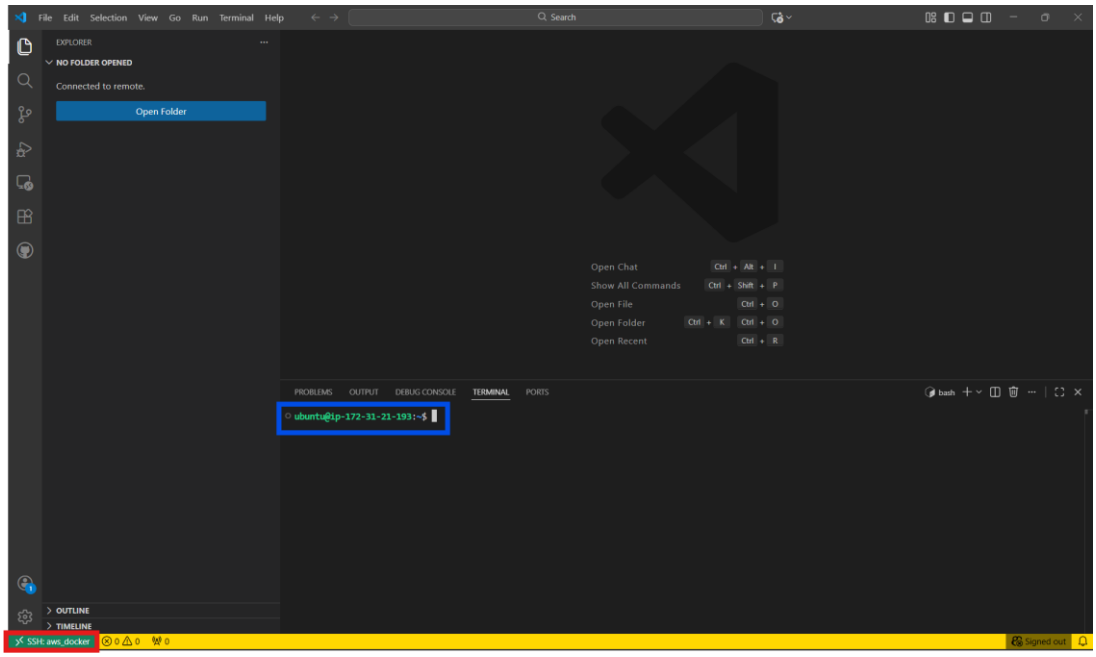
5. Vamos a la extensión “Remote explorer” y seleccionamos “remotes(tunnels/ssh)”. Nos debería detectar automáticamente la configuración añadida en el fichero config del paso 3.d.



6. Asegurándonos de tener el laboratorio arrancado, probamos la conexión seleccionando como máquina remota *Linux*



Si todo va bien, tendremos disponible la terminal conectada por ssh a la instancia EC2 de AWS.



7. Desde la terminal remota, creamos un directorio donde ubicar remotamente nuestros proyectos y repositorios.
    a. Creamos directorio: mkdir -p $HOME/proyectos

```
ubuntu@ip-172-31-21-193:~$ pwd
/home/ubuntu
ubuntu@ip-172-31-21-193:~$ id
uid=1000(ubuntu) gid=1000(ubuntu) groups=1000(ubuntu),4(adm),24(cdrom),27(sudo),30(dip),105(lxd),988(docker)
ubuntu@ip-172-31-21-193:~$ mkdir -p $HOME/proyectos
ubuntu@ip-172-31-21-193:~$ ls -ltr
total 4
drwxrwxr-x 3 ubuntu docker 4096 Jan 31 18:43 proyectos
ubuntu@ip-172-31-21-193:~$ cd proyectos
ubuntu@ip-172-31-21-193:~/proyectos$ git config --global user.email "jose.sala@educa.madrid.org"
ubuntu@ip-172-31-21-193:~/proyectos$ git config --global user.name "Jose Sala"
```

8. Para poder interactuar con el repo remoto de GitHub con nuestras credenciales desde la instancia EC2 de AWS:
    a. Instalaremos el cliente de GitHub: sudo apt install gh

```
ubuntu@ip-172-31-21-193:~$ sudo apt install gh
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
The following NEW packages will be installed:
  gh
0 upgraded, 1 newly installed, 0 to remove and 58 not upgraded.
```

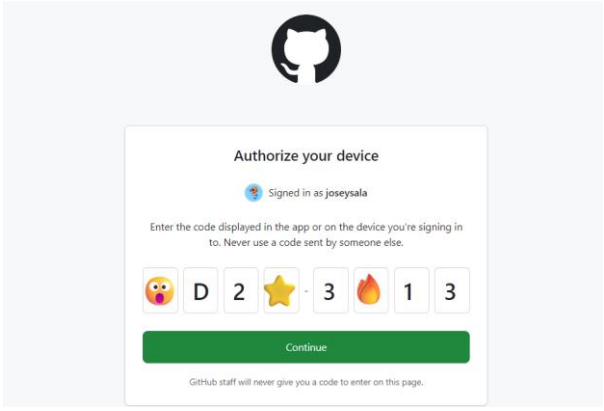
    b. Ejecutamos login desde el cliente de GitHub: gh auth login
    Asegúrate de responder al asistente:
    - *What account?* GitHub.com
    - *Preferred protocol?* **HTTPS**.
    - *Authenticate Git with GitHub credentials?* Yes.

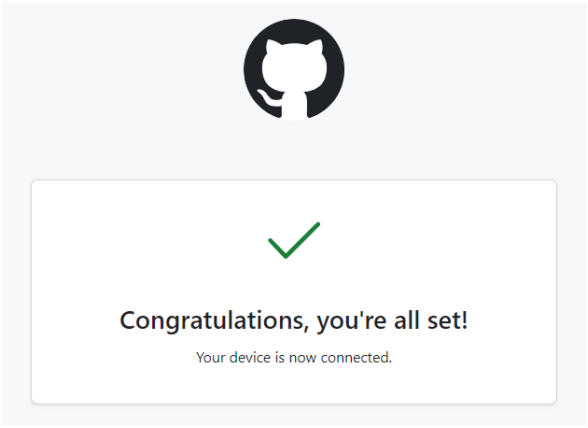
- *How would you like to authenticate?* **Login with a web browser**.

Se abrirá desde tu máquina local el navegador con la url: github.com/login/device y deberás meter ahí el código (si no se abre, hazlo tú). Una vez autorices, la EC2 ya podrá acceder a tu cuenta de github.

```
ubuntu@ip-172-31-21-193:~$ gh auth login
? What account do you want to log into? GitHub.com
? What is your preferred protocol for Git operations on this host? HTTPS
? Authenticate Git with your GitHub credentials? Yes
? How would you like to authenticate GitHub CLI? Login with a web browser

! First copy your one-time code: 😳D2⭐-3🔥13
Press Enter to open github.com in your browser...
```



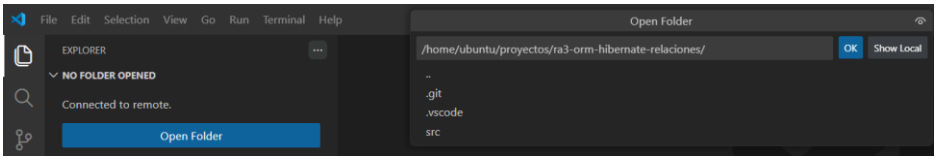


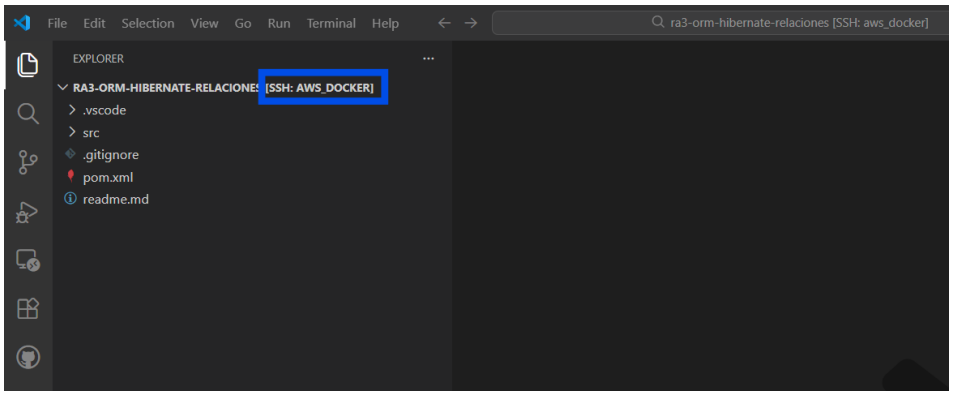

```
Press Enter to open github.com in your browser...
✓ Authentication complete.
- gh config set -h github.com git_protocol https
✓ Configured git protocol
! Authentication credentials saved in plain text
✓ Logged in as joseysala
ubuntu@ip-172-31-21-193:~$
```

9. Ahora ya podemos clonar el repo sobre el que trabajaremos:

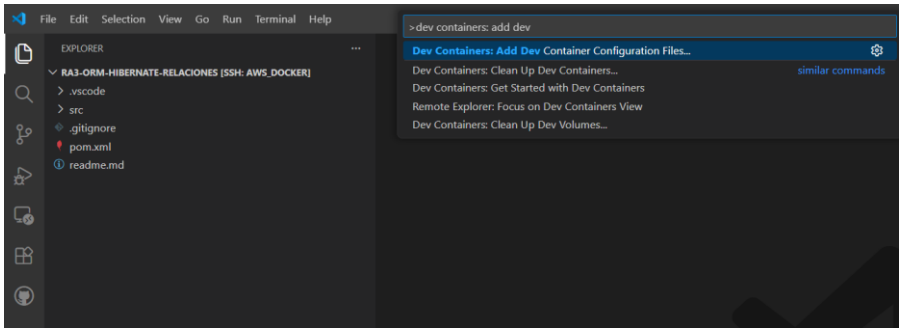
```
ubuntu@ip-172-31-21-193:~$ cd proyectos/
ubuntu@ip-172-31-21-193:~/proyectos$ git clone https://github.com/DAM2-AccesoDatos/ra3-orm-hibernate-relaciones.git
Cloning into 'ra3-orm-hibernate-relaciones'...
remote: Enumerating objects: 129, done.
remote: Counting objects: 100% (129/129), done.
remote: Compressing objects: 100% (59/59), done.
remote: Total 129 (delta 48), reused 118 (delta 37), pack-reused 0 (from 0)
Receiving objects: 100% (129/129), 33.31 KiB | 4.16 MiB/s, done.
Resolving deltas: 100% (48/48), done.
ubuntu@ip-172-31-21-193:~/proyectos$
```

10. Estando contectados a la EC2 mediante “remote explorer”, vamos a hacer que VSCode local "salte" de nuestro equipo local a la EC2 y luego "se meta" dentro del contenedor donde desarrollaremos sobre el proyecto clonado.
    a. Desde File -> open folder, seleccionamos la ruta remota donde está nuestro repo clonado:



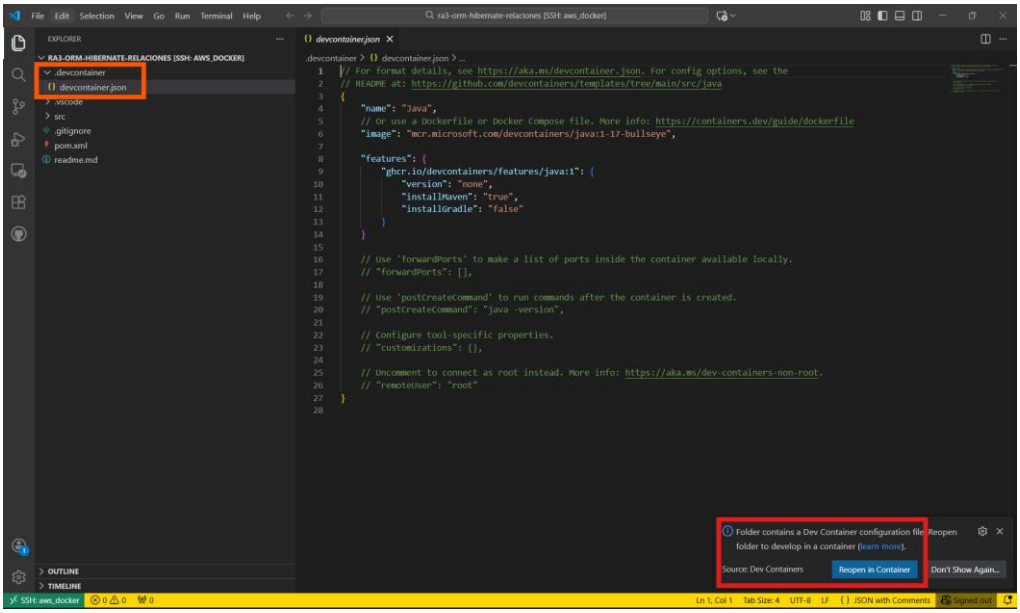


    b. Ctrl + May + P y escribimos "Dev Containers: Add Dev Container Configuration Files..."



    c. Seleccionamos la plantilla existentes de **Java devcontainers + versión: 17 bullseye + Install Maven.** Por ahora nada más…

Automáticamente se crea un directorio **.devcontainer** con un archivo **devcontainer.json**. Y VSCode al detectarlo nos da la opción cuando queramos de **"Reopen in Container"**.



d. Editaremos el fichero json para añadir las extensiones y usuario remoto necesario para trabajar con VScode:

```
"remoteUser": "vscode",
"customizations": {
   "vscode": {
      "extensions": [
         "vscjava.vscode-java-pack",
         "vscjava.vscode-hibernate",
         "cweijan.vscode-database-client2"

      ]
   }
}
```

devcontainer.json

.devcontainer > devcontainer.json > ...

```
// For format details, see https://aka.ms/devcontainer.json. For config options, see the
// README at: https://github.com/devcontainers/templates/tree/main/src/java
{
    "name": "Java",
    // Or use a Dockerfile or Docker Compose file. More info: https://containers.dev/guide/dockerfile
    "image": "mcr.microsoft.com/devcontainers/java:1-17-bullseye",

    "features": {
        "ghcr.io/devcontainers/features/java:1": {
            "version": "none",
            "installMaven": "true",
            "installGradle": "false"
        }
    },
    "remoteUser": "vscode",
    "customizations": {
        "vscode": {
            "extensions": [
                "vscjava.vscode-java-pack",
                "vscjava.vscode-hibernate",
                "cweijan.vscode-database-client2"
            ]
        }
    }
}
```

e. Ahora arrancamos el dev container, Ctrl + May + P y escribimos "Dev Containers: Reopen in Container", con ello sucederá lo siguiente de forma transpartente para nosotros:

1. El Salto de la interfaz (Client-Server)
VS Code detecta que estás en una máquina remota y entiende que no debe buscar Docker en tu laptop, sino en el **Docker Engine de la EC2**. Tu VS Code local se convierte en una simple "pantalla" (Client), mientras que toda la lógica se traslada al servidor.

2. Lectura del Plano (devcontainer.json)
Busca la carpeta .devcontainer y lee las instrucciones:

- **¿Qué imagen uso?** (ej. Java 21).
- **¿Qué puertos abro?** (ej. 8080 para tu app, el puerto de H2).
- **¿Qué extensiones instalo?** (Hibernate, Database Client).

3. Construcción del Entorno (The Build)
Si es la primera vez, verás que se abre una terminal llamada **"Starting Dev Container"**. En la EC2 ocurre lo siguiente:

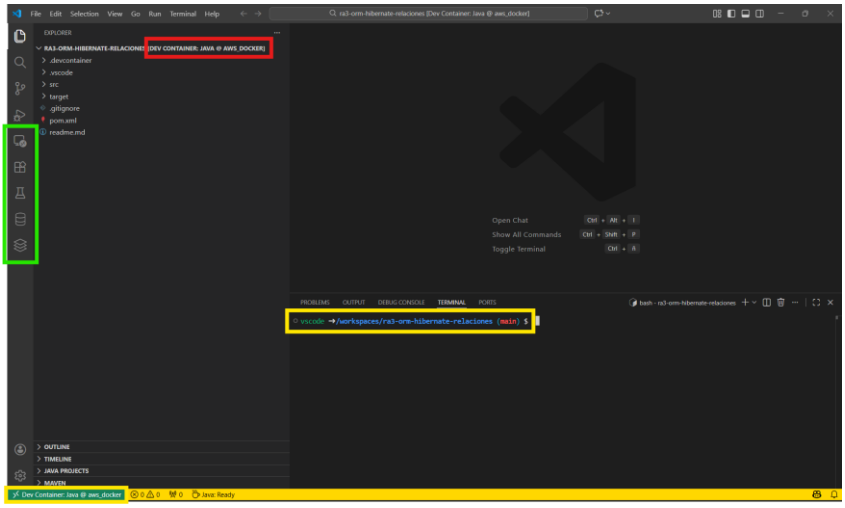
- **Pull:** Descarga la imagen base de Java desde el registro de Microsoft.
- **Features:** Si añadiste el CLI de GitHub o Docker-in-Docker, los instala en ese momento sobre la imagen.
- **Mounting:** Aquí ocurre la magia: **mapea la carpeta de la EC2 donde clonaste el código** a una ruta interna del contenedor (normalmente /workspaces/nombre-del-repo).

4. Inyección del "VS Code Server"
Este es el paso secreto. VS Code descarga e instala un pequeño motor binario **dentro del contenedor**.

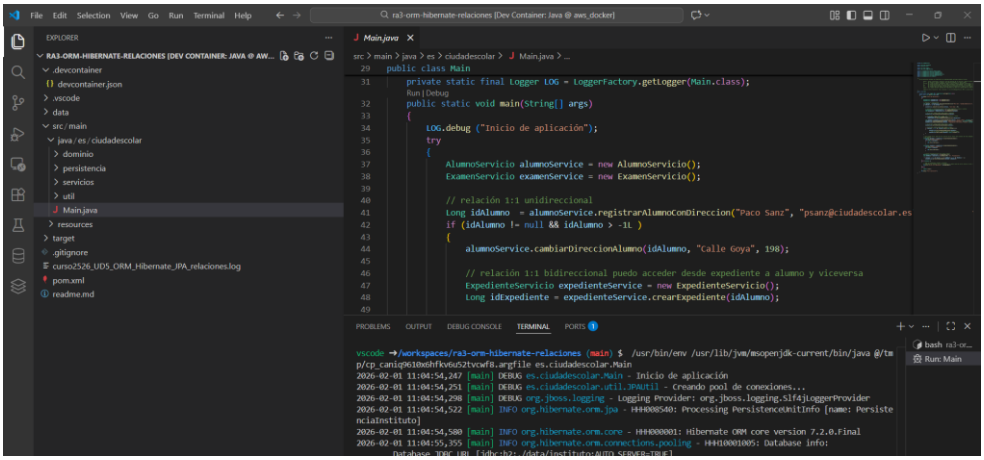
- Este servidor es el que realmente ejecuta el compilador de Java, el Language Server (IntelliSense) y las extensiones.
- Esto garantiza que, aunque tu laptop sea vieja o no tenga Java, el autocompletado sea instantáneo porque se ejecuta justo al lado del código.

5. Configuración del Terminal y Usuario

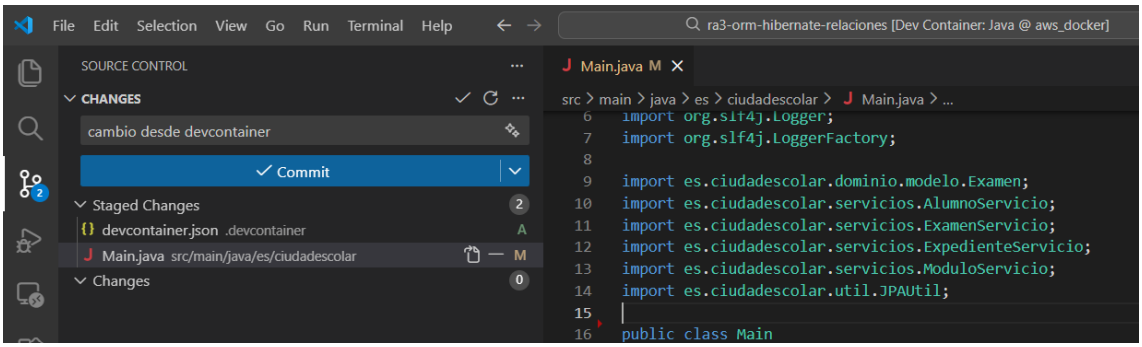
Cambia el contexto de tu terminal. Al terminar, la terminal que ves en VS Code ya no dice ubuntu@ip-172-x-x-x, sino algo como vscode@hash-del-contenedor. **Ahora cualquier comando que escribas (como mvn clean install) se ejecuta en el entorno aislado con Java.**



11. Ya podemos realizar los cambios que queramos en nuestro repo y ejecutar dentro del contenedor:

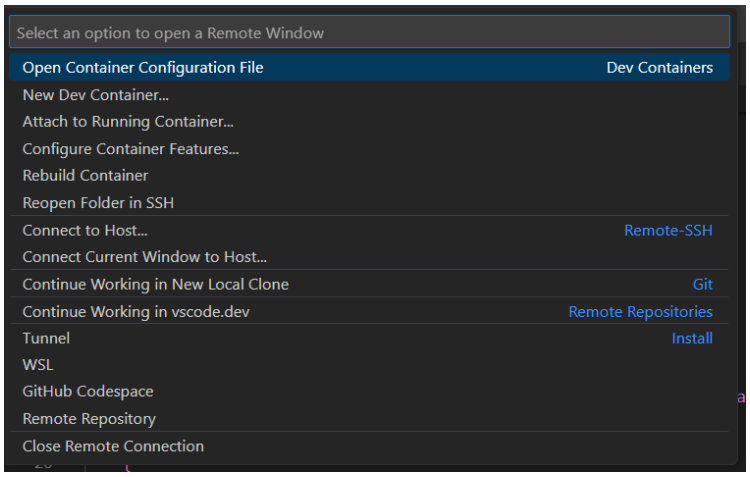


12. Para poder subir los cambios en nuestro repo, debemos habilitar git en settings git.enabled. Automáticamente se visualizará en “source control” de VSCode los cambios. Podremos hacer commit y subirlos al repo remoto (si tenemos permisos).

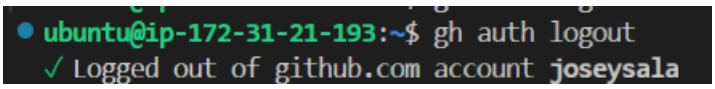


13. Haciendo clic en el incono 

, podremos cerrar la conexión con el contedor seleccionando “reopen folder in SSH”. De esa forma estaremos en el repo clonado en la máquina remota (pero fuera del contenedor)



14. Si queremos quitar permisos a la instancia EC2 de nuestra cuenta de Github podemos desde la máquina EC2 hacer el logout.



15. Haciendo clic en el icono 

podremos finalmente cerrar la conexión remota con la instancia EC2 y retornar al VSCode local, seleccionando “close remote connection”