Cloud Design

La implementación se hizo real en la nube a través de AWS, en el archivo por nombre “Arquitecture Diagram.pdf” se encuentra como esta estructurado el prototipo que se hizo para esta prueba con sus componentes.

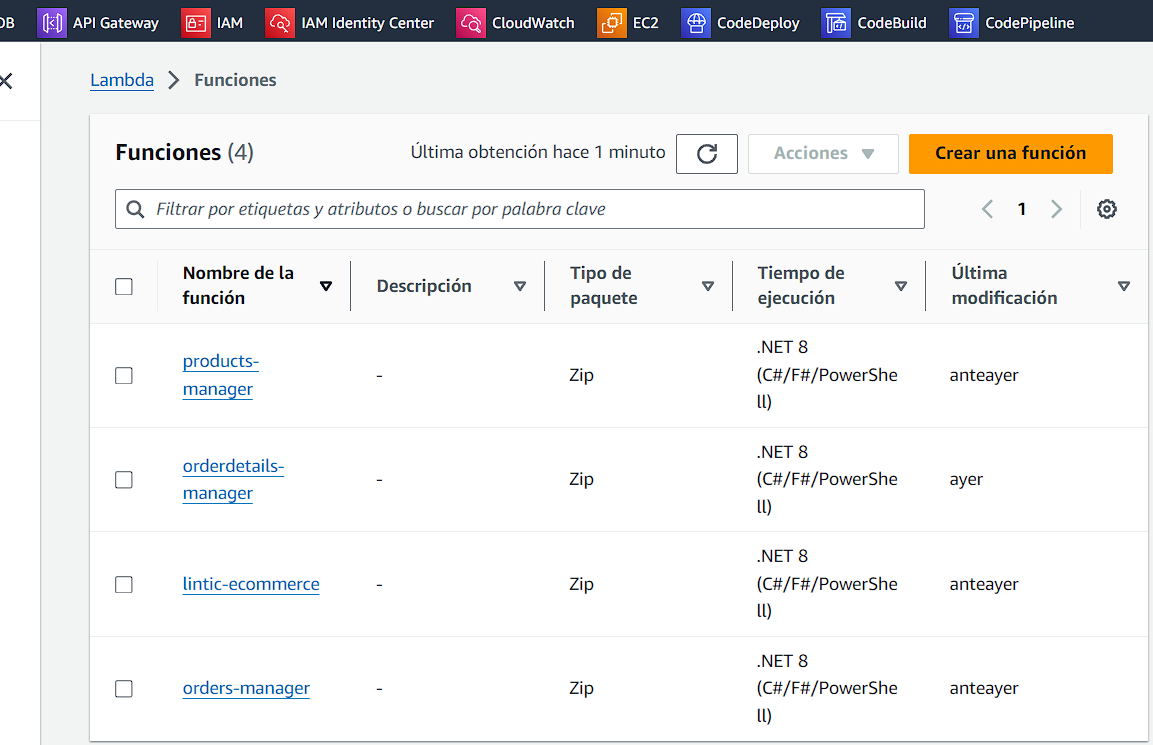
Basado en ellos se implementó de acuerdo a los siguientes pasos:  
  
1. Se Identifico los servicios de AWS necesarios para llevar la implementación, estos fueron

* **Lambda :** para la configuración de la capa de “Repositories” que permita la interacción con la base de datos.
* **DynamoDB:** se utilizo como base de datos no relacional, en este caso se escogio para llevar todo a una arquitectura serveless.
* **API Gateway:** este se empleo para administrar de manera mas eficiente el consumo de las funciones lambda y por seguridad ya que interactúan con la Base de datos.
* **S3:** Se empleo para almacenar los artefactos que se generan en el deploy y sean subidos a la instancia para su despliegue en ambiente sea de pruebas o producción.
* **EC2:** Como instancia para la escalabilidad en la nube de la aplicación, permitiendo lanzar varias instancias (pods) simultáneos, con las configuraciones del hardware y sistema operativo necesario y por su flexibilidad. Se creo instancias bajo Linux ya que .NET 8 nos permite su instalación en este tipo de SO.
* **CodeDeploy, CodeBuild, CodePipeline:** estos se emplearon de la mano para la canalización del CI/DI en la ejecución de la aplicación de prueba en EC2, se integró a GitHub y S3.
* **SQS(Simple Notification Service):** se utilizo La cola SQS recibe y almacena el mensaje temporalmente, hay una lambda que procesa el mensaje y lo guarda en la tabla Approval y actualiza la de TodoTask.
* **GitHub:** se utilizo como repositorio y versionado de nuestro código fuente para las API, con sus respectivos microcommits y manejo de branchs.
* **Microservicios:** se utilizó la versión más actualizada de .NET 8, se implementaron diferentes métodos API Rest para la integración con el FrontEnd.
* **CloudWatch:** para el manejo de logs de toda la integración.
* **IAM:** Para el manejo de los usuarios y los roles, permisos, grupos para ejecución e implementación dependiendo de los servicios a utilizar.

2. Para las funciones de la Lambda se creó el repositorio baso .NET core 8:

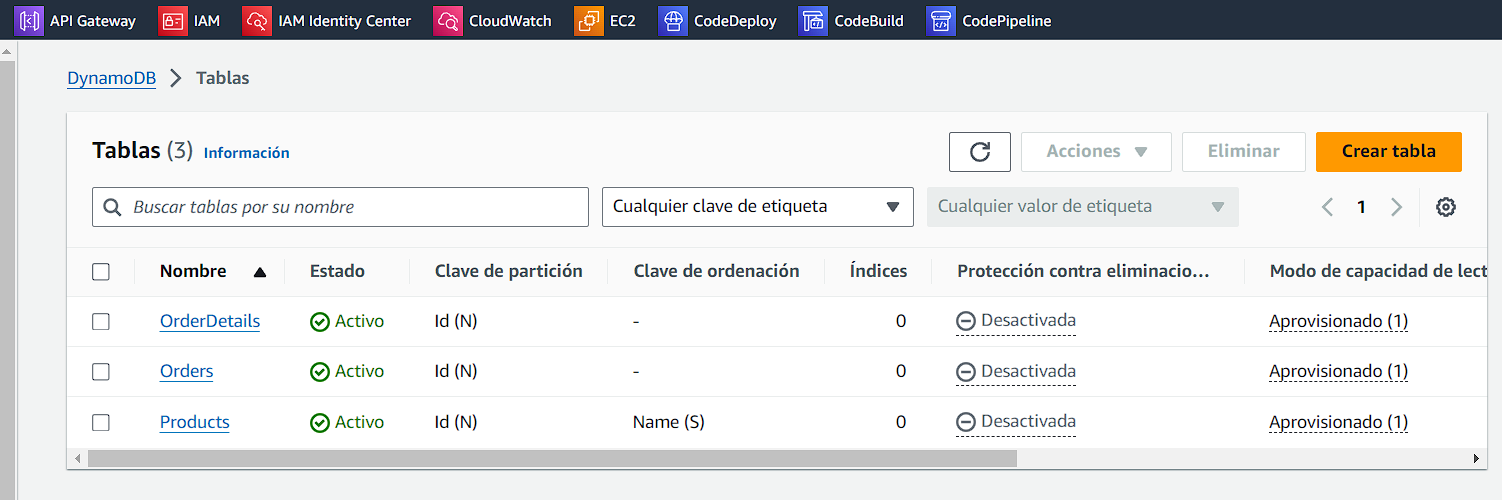
* <https://github.com/jescorcia18/eCommerceAWS/tree/master>

y estas se publicaron para que cargaran a nuestra configuración:

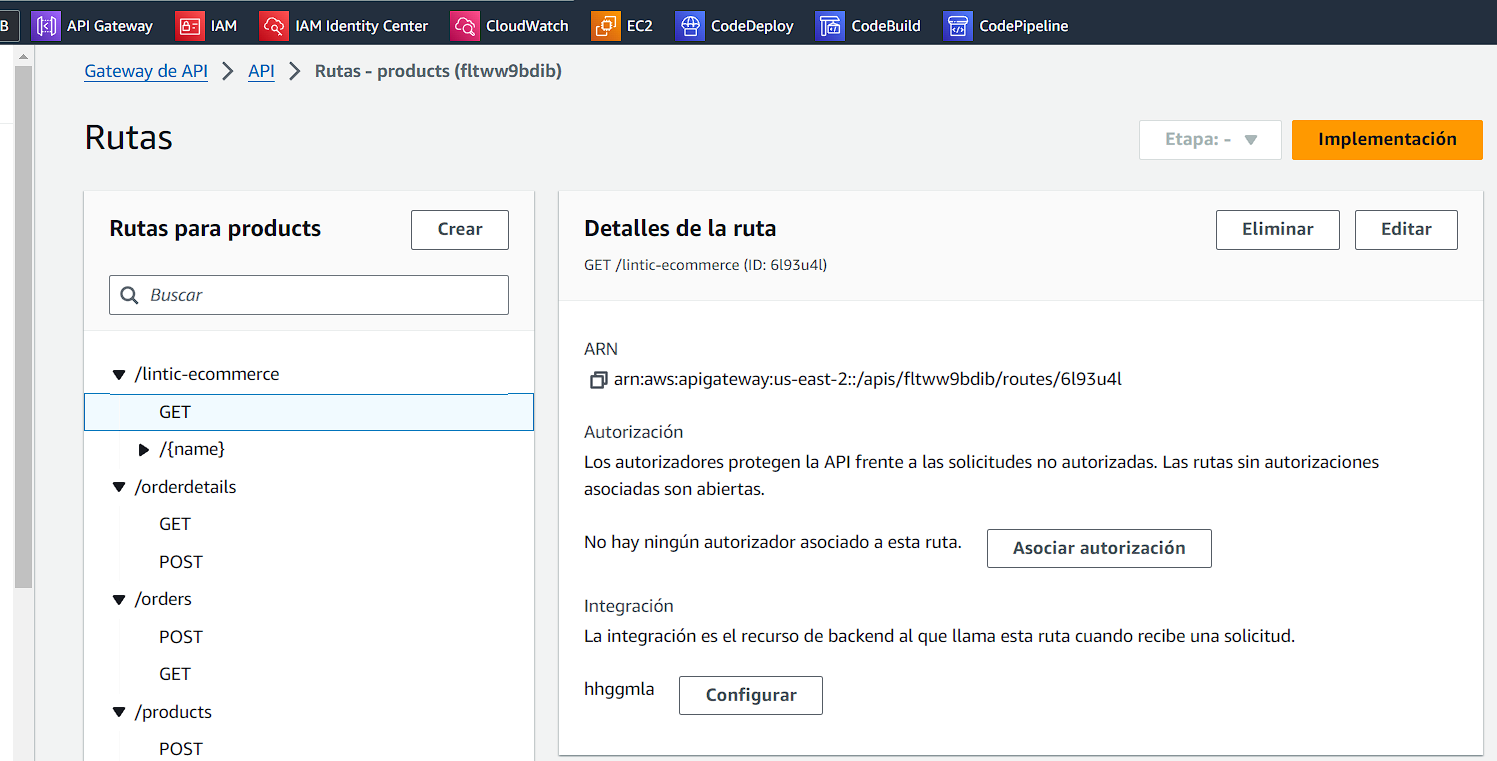


Las Lambda Fuctions actualmente publicadas:

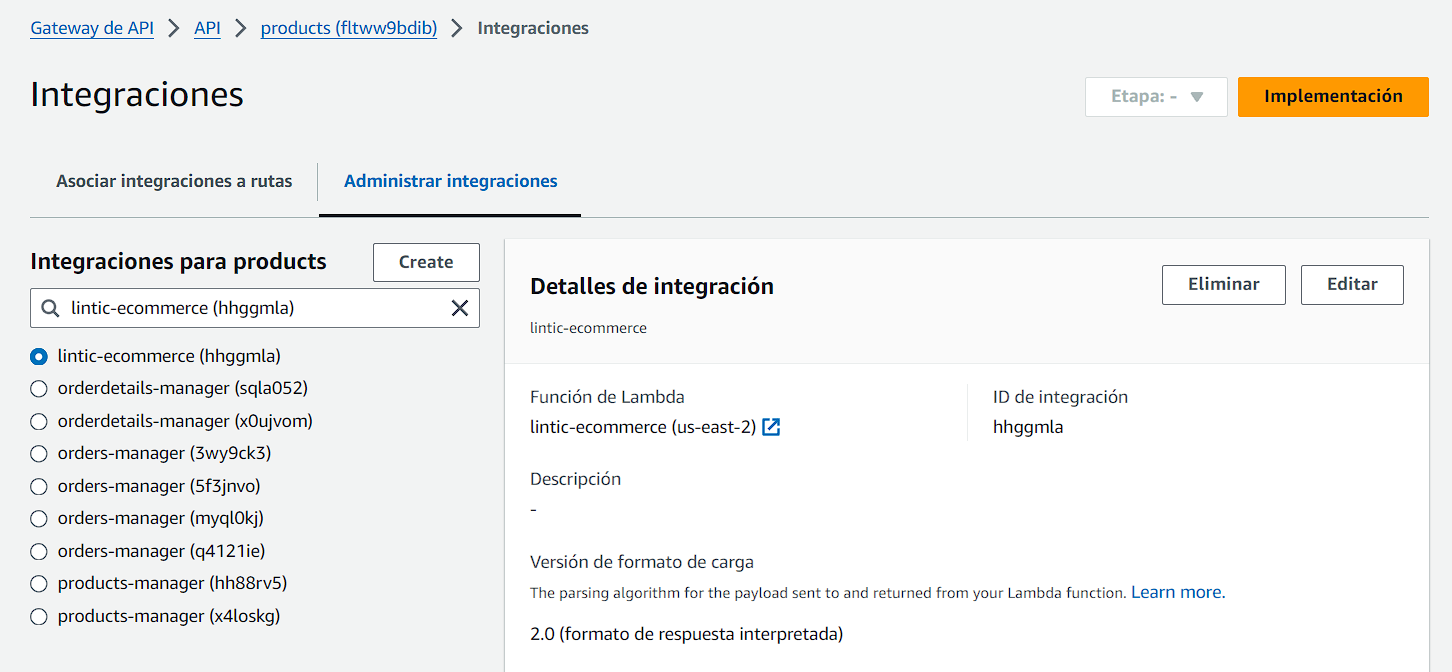
* **Products:** <https://fltww9bdib.execute-api.us-east-2.amazonaws.com/products>
* **Order:** <https://fltww9bdib.execute-api.us-east-2.amazonaws.com/orders>
* **OrderDetails:** <https://fltww9bdib.execute-api.us-east-2.amazonaws.com/orderdetails>

3. En DynamoDB se crearon las tablas:

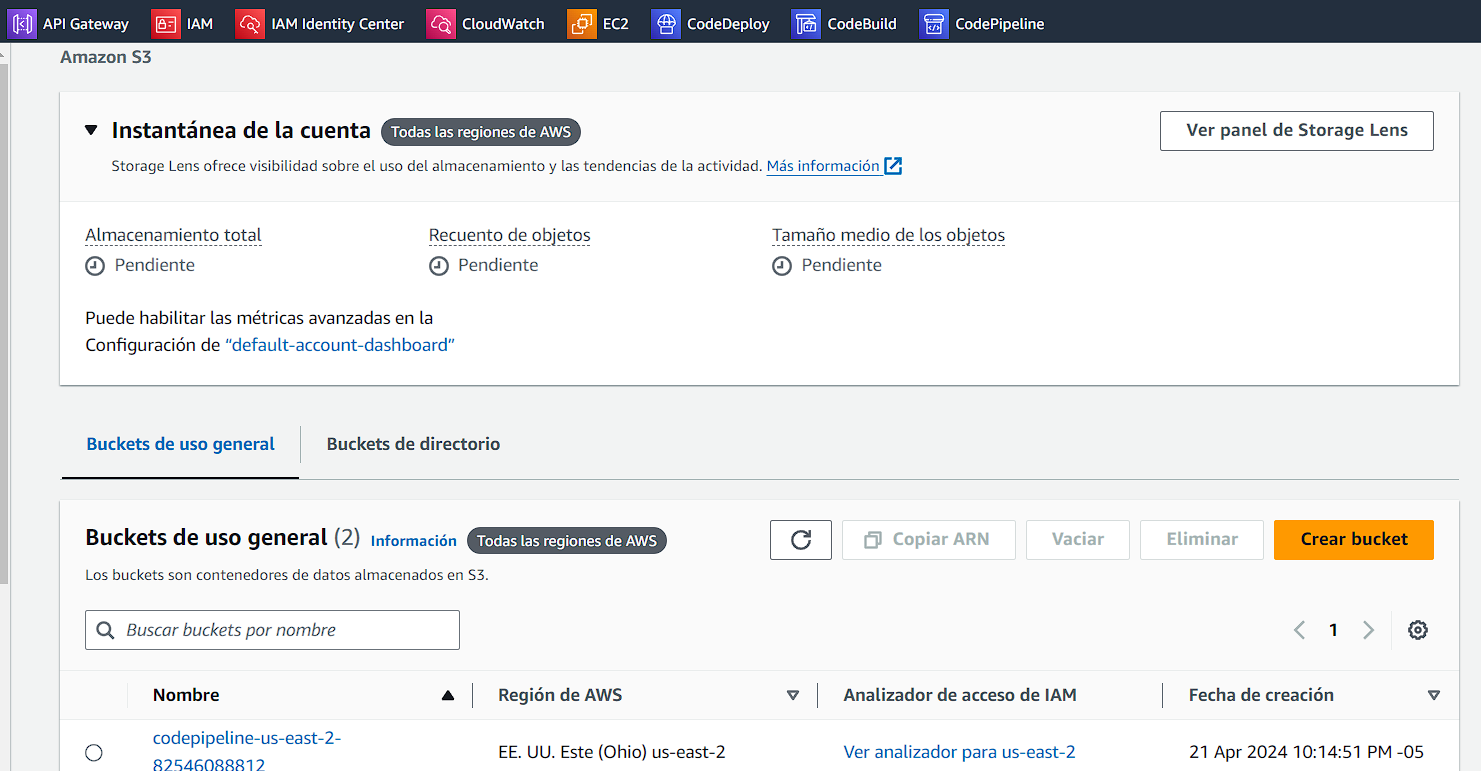
4. En API Gateway se parametrizaron las siguientes rutas:



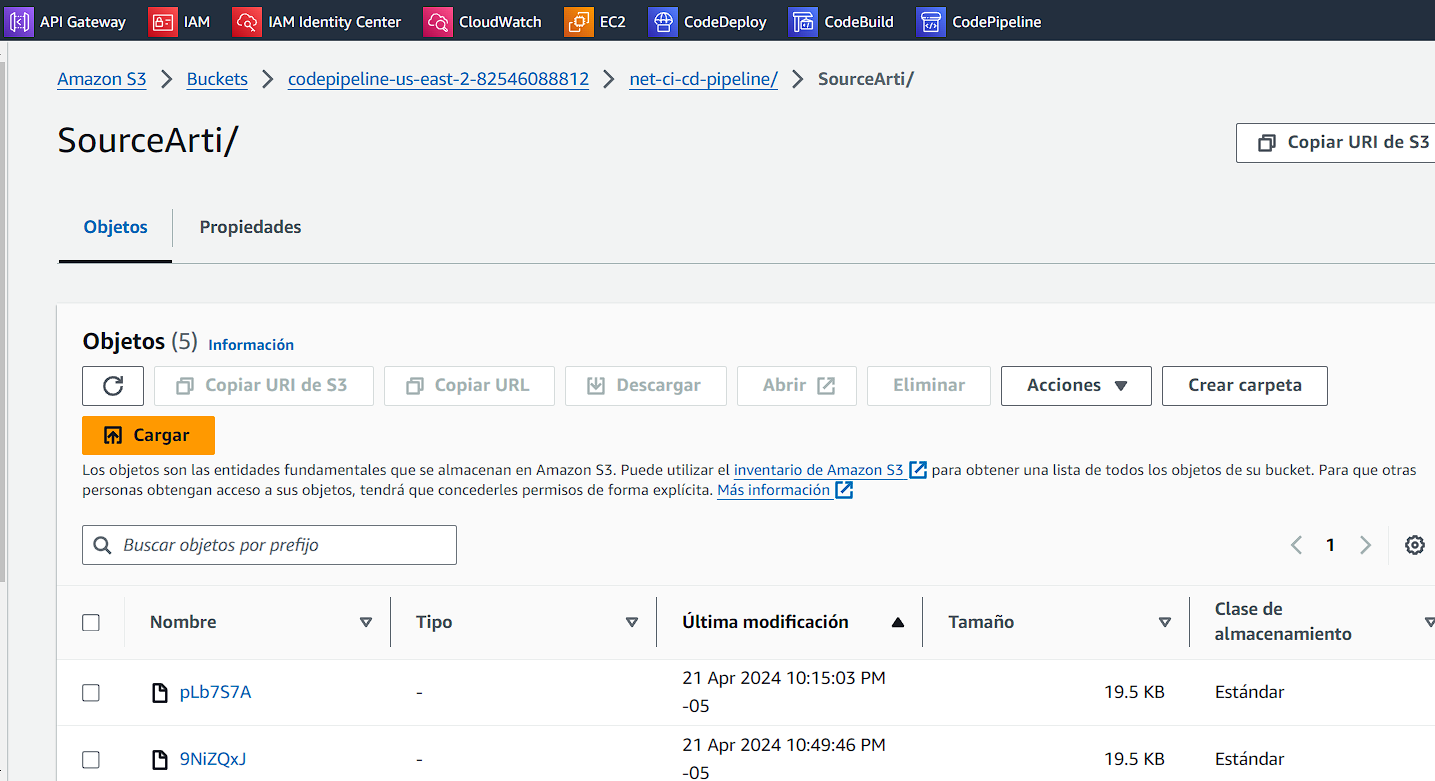
Y se les asocia las Lambdas respectivas:



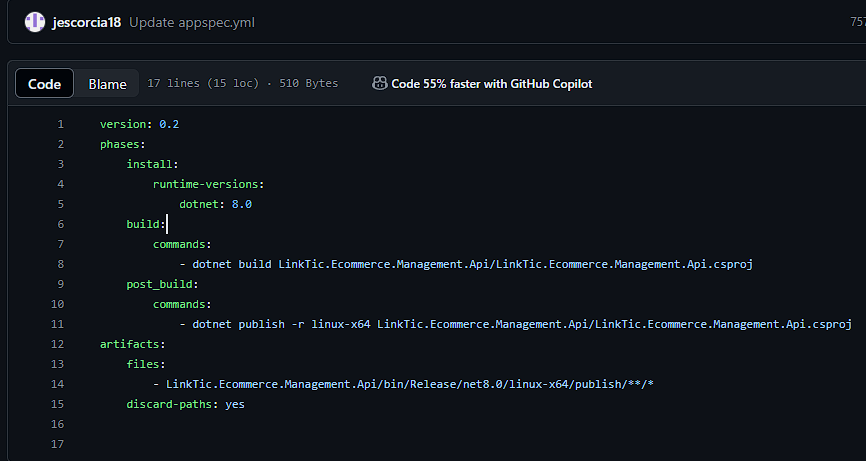
5. Se creo un Bucket en S3 para los artefactos:



En la siguiente imagen vemos las carpetas almacenadas con los diferentes artefactos:



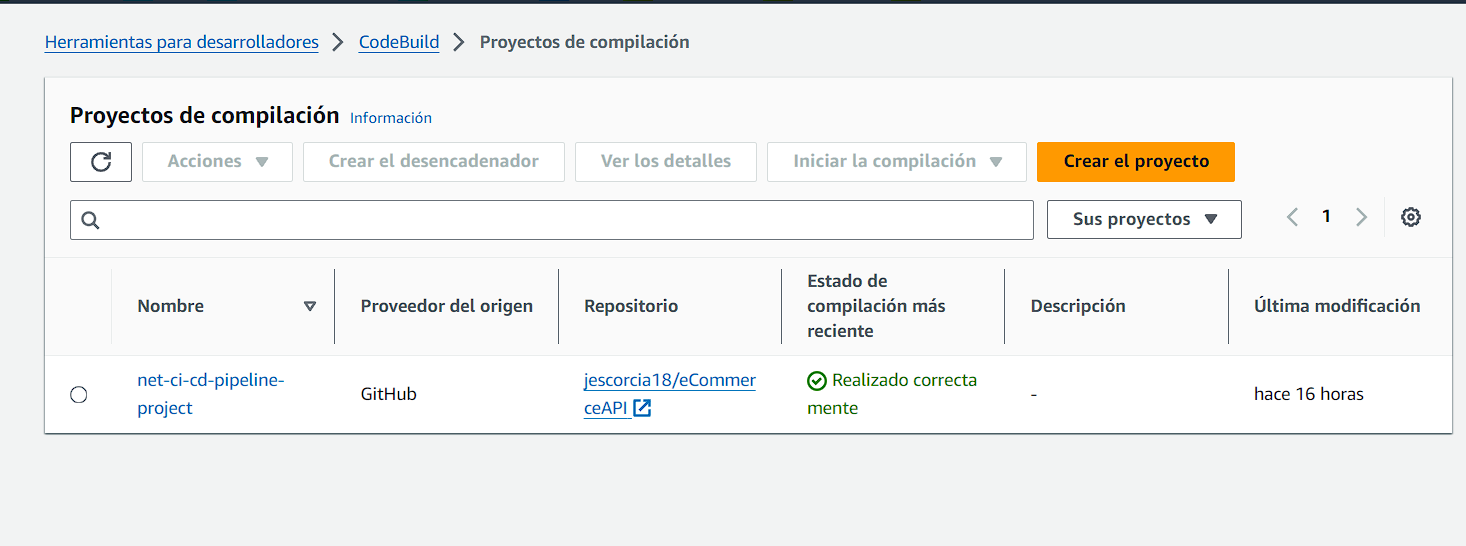
6. Creamos y configuramos nuestro archivo “buildspec.yml” que en mi caso yo lo llame “appspec.yml”:



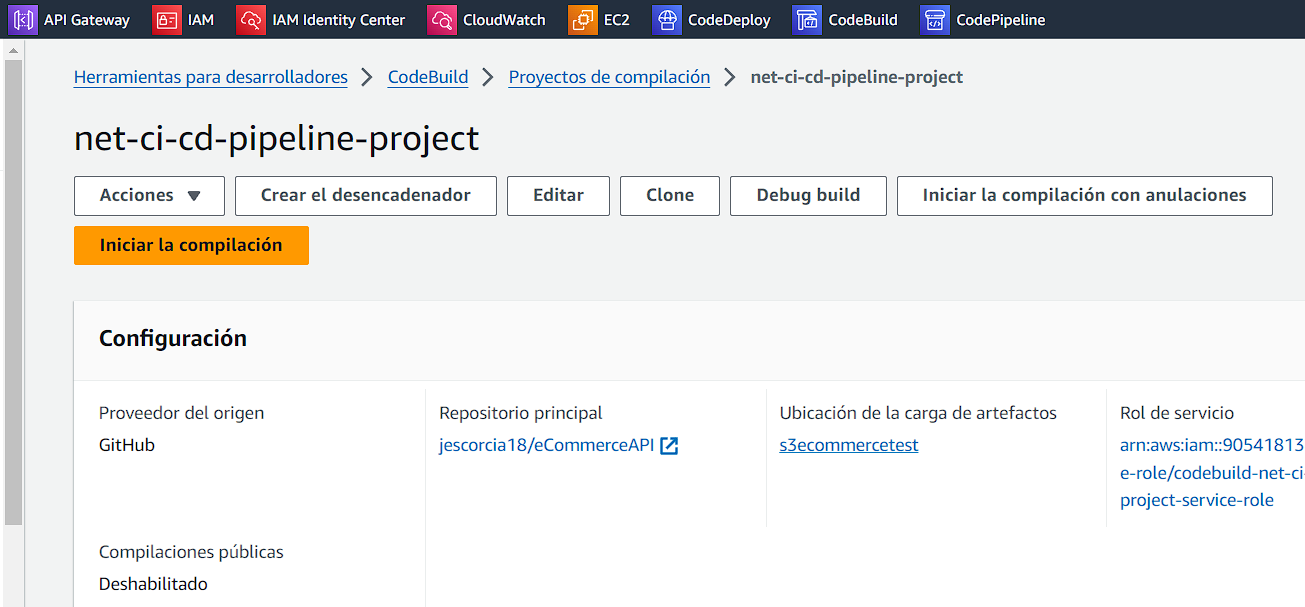
Esto permite que se ajecute el build en el proyecto, publicando la aplicación de forma autónoma e independiente del tiempo ejecutándose para Linux, con un entorno instalado de .NET en la instancia de EC2. Este se sube a la raíz en mi caso del proyecto.

7. Se crea y se configura la compilación para el proyecto y utilizamos AWS CodeBuild:

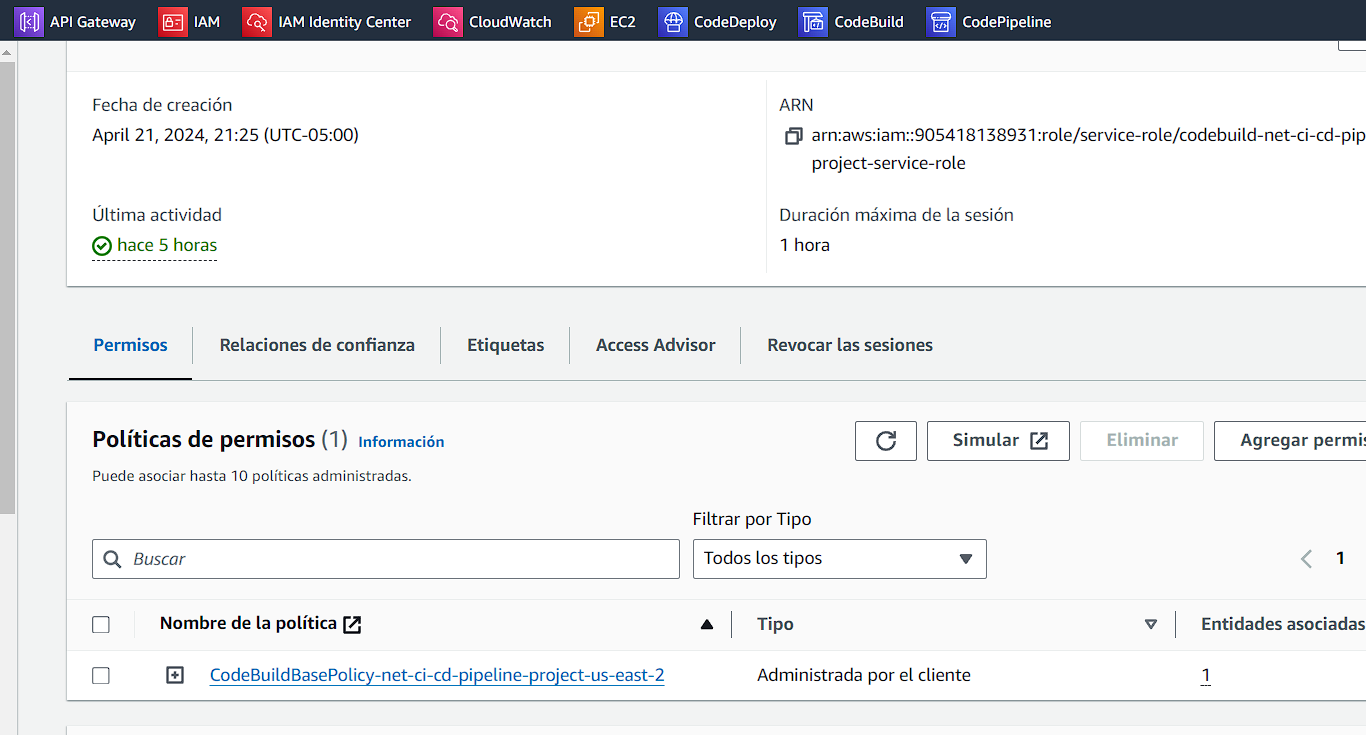
* se crea un nuevo proyecto



* se integra y se asocia nuestro proveedor de GitHub

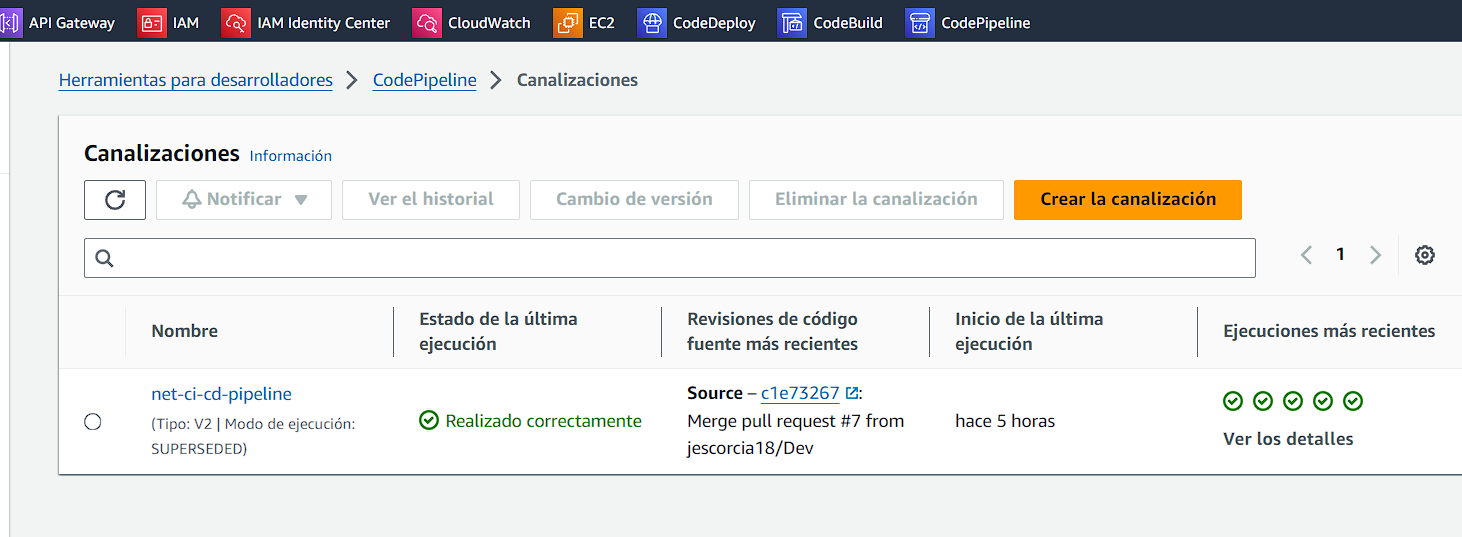


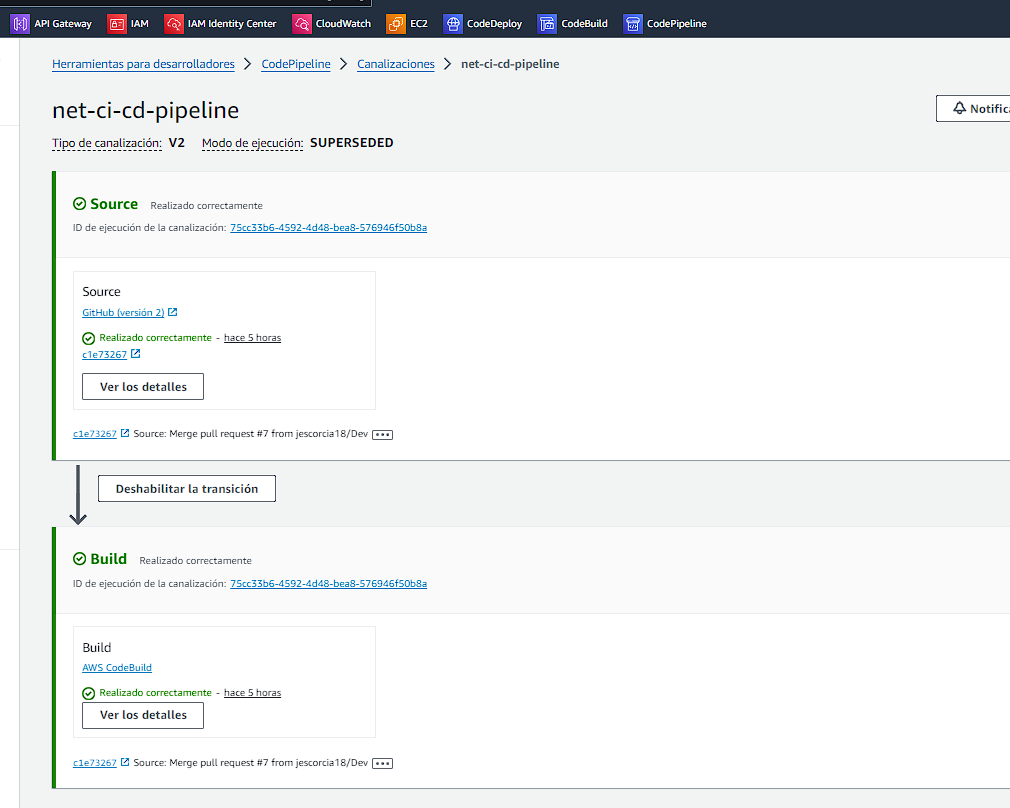
* Se debe crear un Rol con permisos como se muestra en la imagen posterior para el manejo de los 4 servicios integrados como son el S3, el IcloudWatch, codeBuild y el SSM Message.





8. Se crea las configuraciones para CI/DI pipeline en una instancia EC2 con el servicio CodePipeline, se integra para la autogeneración autómatica con GitHuh en la rama especifica para este caso “main”



Acá vemos ya configurado la ejecución automática cuando se realiza los cambios y como este los procesa:  
  


9. Se crea el CodeDeploy que viene de la mano de CodePipeline para la integración.

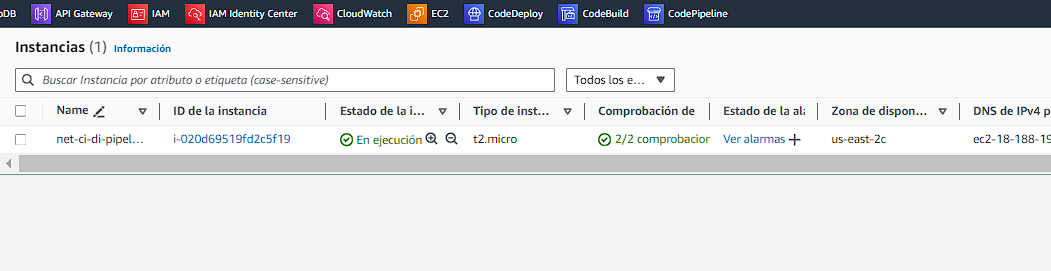
* En el CodeDeploy se crea una Aplicación en plataforma EC2

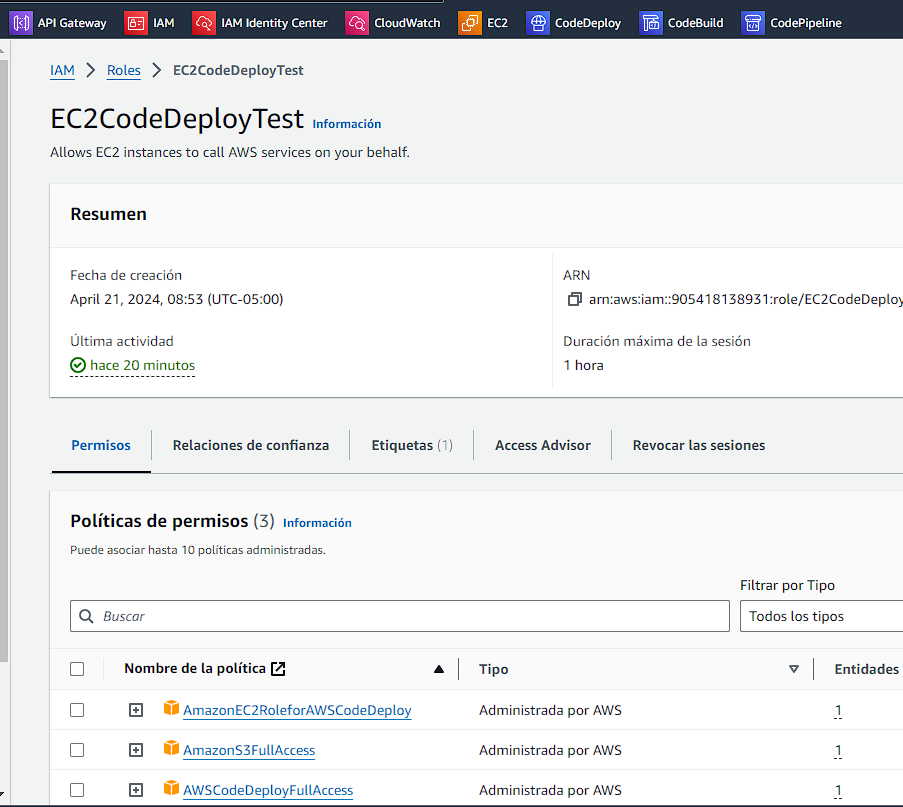


* Se agrega un grupo de Implementaciones y se implementa:

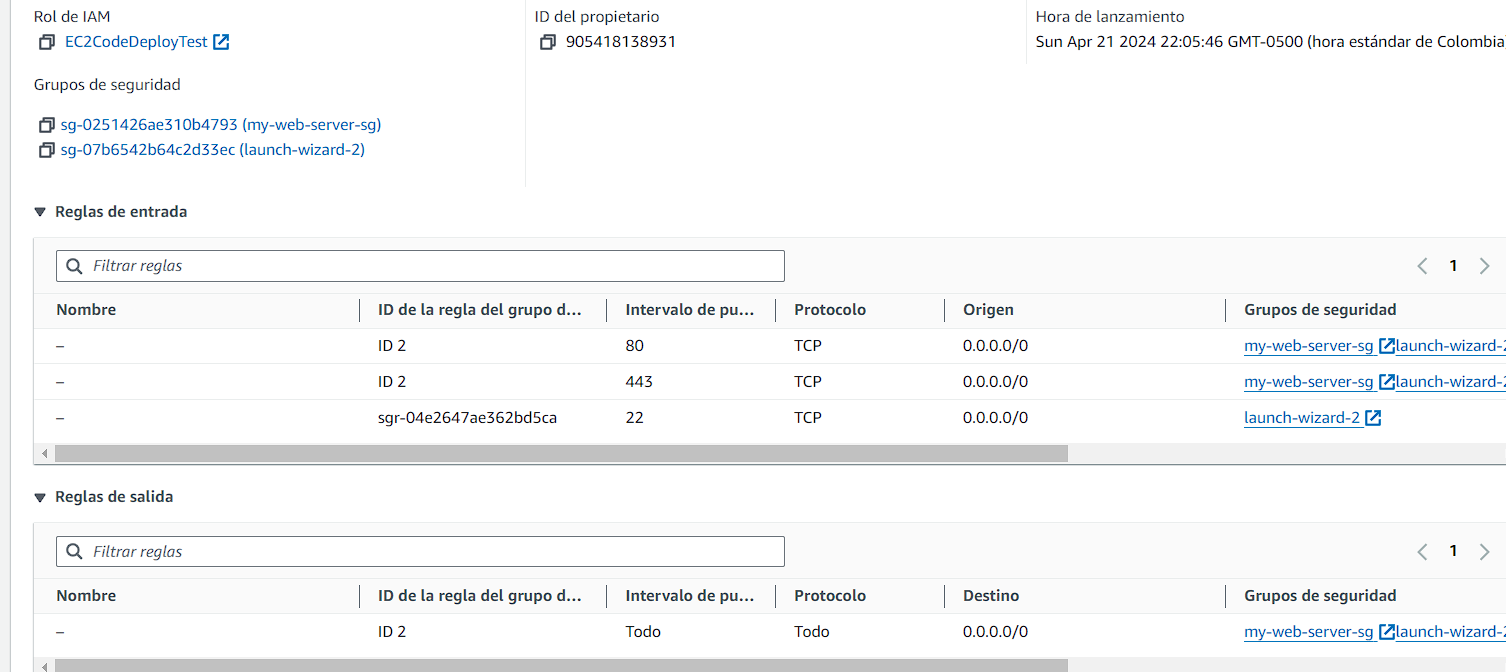


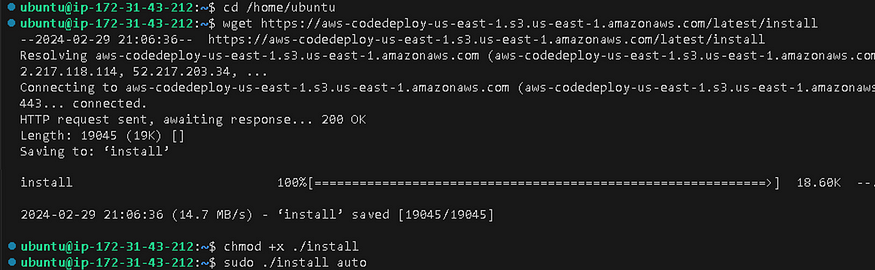
10. se crea la instancia en EC2, junto a un rol para el deploy que ya fue creado con los permisos para poder ejecutar



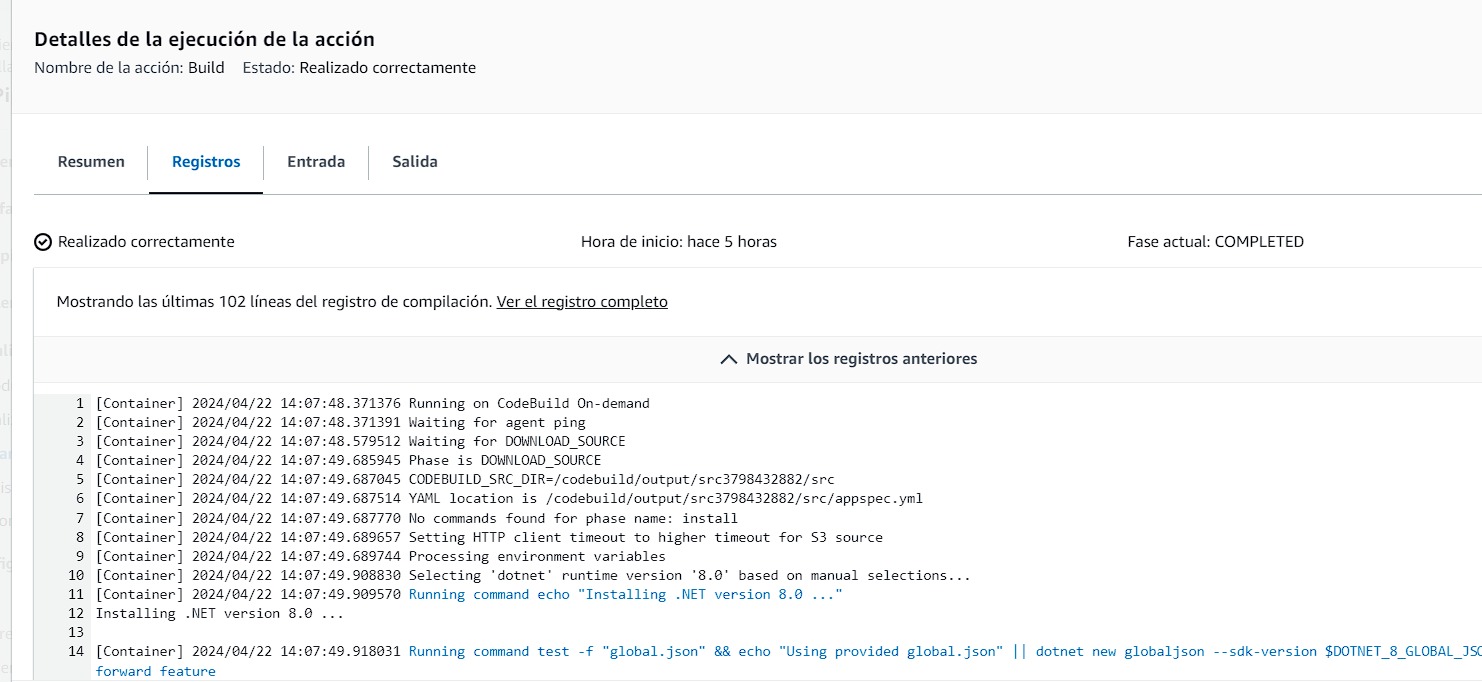
Aca vemos el rol:

12. Se configura las reglas de entrada y salida, en este caso pusimos todo pero se debe permitir de acuerdo a las IP que accedan o consuman esta instancia.



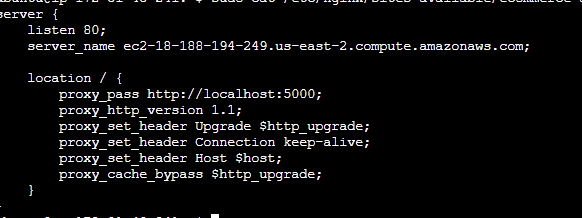
13. Abrimos Consola de nuestra instancia en EC2 para instalar el “Agente CodeDeploy”:  
  


12. Se crea el pipeline, para que pueda ejecutarse automáticamente y podemos ver lo Logs por medio del services IcloudWatch:

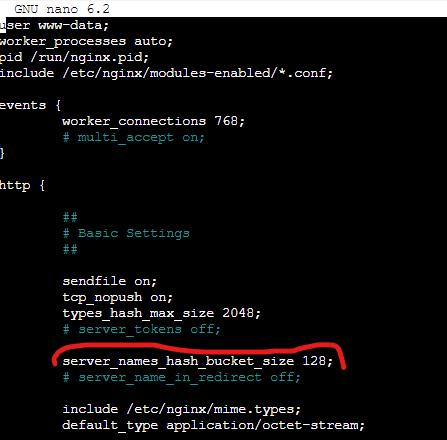


13. Dependiendo el caso para que la aplicación entre en ejecución se debe:

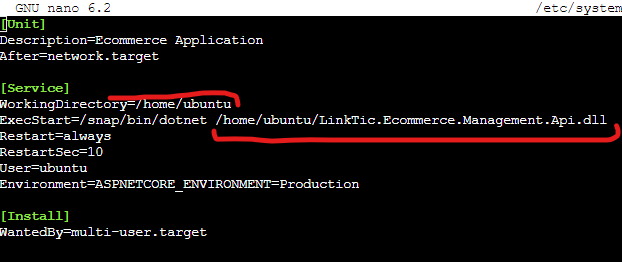
* Instalar Nginx: como servidor web en nuestro servidor Linux configurado en el EC2
  + Sudo apt install nginx
* Verificamos que su estado este activo o corriendo.
  + Systemctl status nginx
* Inicializamos nginx sino lo esta
  + sudo service nginx start
* configuramos Nginx como servidor web inverso



* Reiniciamos Nginx:
  + sudo systemctl restart nginx
* Nos traemos el artefacto de S3 y lo decomprimimos
  + aws s3 cp s3://nombre-del-bucket/ruta/al/artefacto.zip .
  + unzip artefacto.zip
* Posterior probamos que sea el artefacto correcto y lo corremos:
  + dotnet LinkTic.Ecommerce.Management.Api.dll
* En la configuración del archivo .confi de nginx es ideal deshabilitar la línea del tamaño del hash\_bucket y cambiarle el valor predeterminado por 128:
  + sudo nano /etc/nginx/nginx.conf

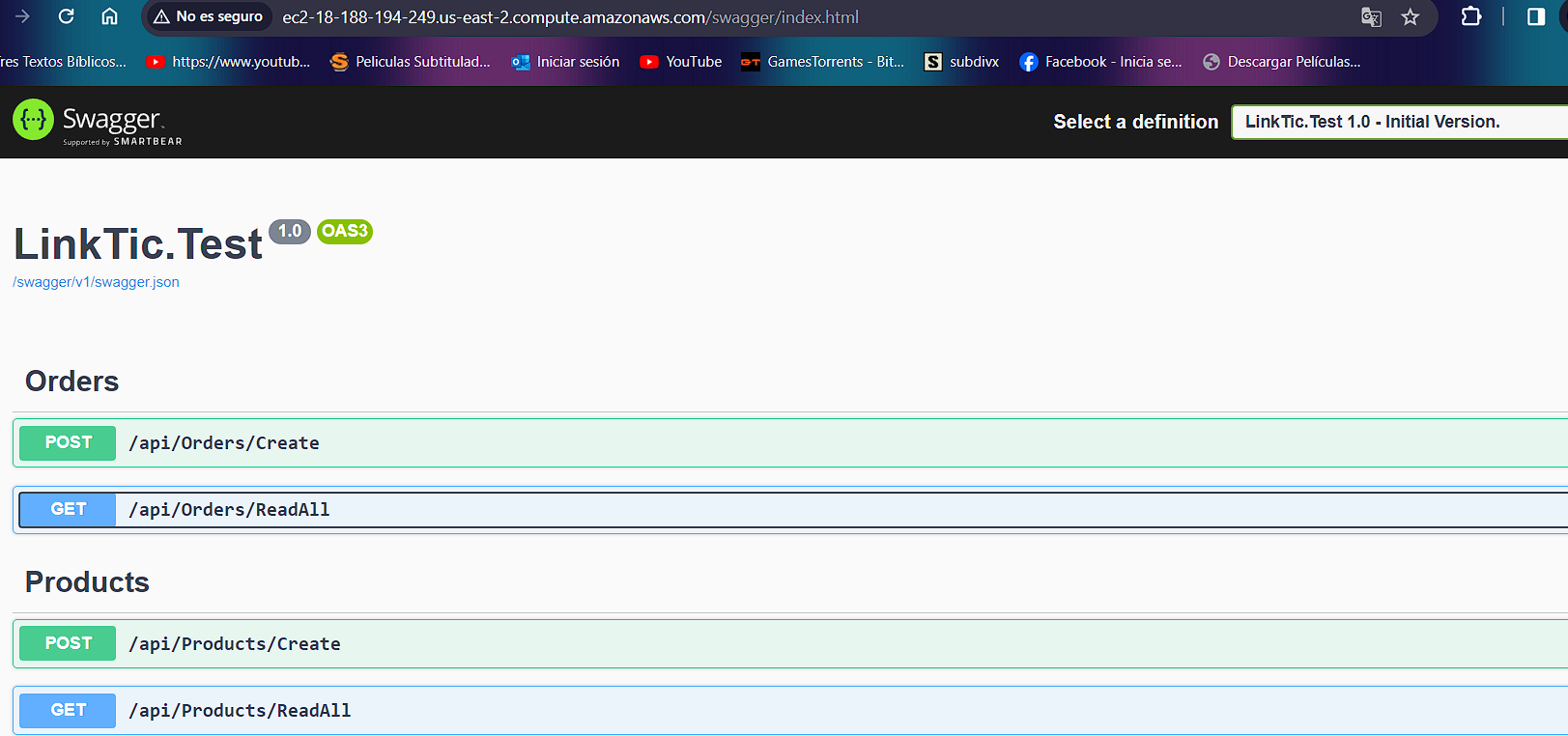


* Se crea un archivo de servicio de systemd para la aplicación en .NET Core y se ejecute automática
  + sudo nano /etc/systemd/system/dotnet-ecommerce.service
  + es muy importante tener en cuenta en donde quedaron los archivos descomprimidos del artefacto y el usuario actual de la instancia que trajimos del S3:





* y queda desplegado en nuestro servidor inverso Nginx nuestra y de manera pública:
  + <http://ec2-18-188-194-249.us-east-2.compute.amazonaws.com/swagger/index.html>



**NOTA: Repositorios GitHub**

-API : <https://github.com/jescorcia18/eCommerceAPI>

-LAMBDA: <https://github.com/jescorcia18/eCommerceAWS/tree/master>