Tarek Ben Hamdouch

AUTOMATITZACIÓ DEL DESPLEGAMENT DE MODELS D'INTEL·LIGÈNCIA ARTIFICIAL FETS AMB MONAI A AMAZON WEB SERVICES (AWS)

TREBALL DE FI DE GRAU

Dirigit per Jordi Massaguer Pla

Doble grau en Enginyeria Informàtica i en Biotecnologia

UNIVERSITAT ROVIRA I VIRGILI

Tarragona

2024

Resum.

Aquest projecte va sorgir davant la necessitat de facilitar el desplegament automatitzat de models d'intel·ligència artificial per a la inferència en imatges mèdiques utilitzant serveis al núvol d'Amazon Web Services (AWS). La intel·ligència artificial està transformant el camp de la medicina, millorant l'eficiència i precisió en el diagnòstic i tractament de malalties. En aquest context, MONAI (Medical Open Network for AI) proporciona una plataforma de codi obert per al desenvolupament de models d'IA en imatges mèdiques, però portar aquests models a producció continua sent un repte significatiu.

Per realitzar el projecte s'han utilitzat GitHub Actions per automatitzar i gestionar els serveis d'AWS, permetent empaquetar els models i executar-los sense necessitat d'intervenció humana. A més a més, s'ha configurat el sistema perquè faci la inferència de manera autònoma cada vegada que es puja una imatge a Amazon S3.

Aquesta automatització redueix el temps necessari per portar un model des de la fase de desenvolupament fins a la producció, garantint consistència i escalabilitat. El projecte demostra l'eficàcia i la viabilitat d'aquesta prova de concepte, permetent als investigadors centrar-se en l'optimització dels models en lloc de les tasques de desplegament, millorant així l'eficiència i reduint els costos operatius.

Resumen.

Este proyecto surgió ante la necesidad de facilitar el despliegue automatizado de modelos de inteligencia artificial para la inferencia en imágenes médicas utilizando servicios en la nube de Amazon Web Services (AWS). La inteligencia artificial está transformando el campo de la medicina, mejorando la eficiencia y precisión en el diagnóstico y tratamiento de enfermedades. En este contexto, MONAI (Medical Open Network for AI) proporciona una plataforma de código abierto para el desarrollo de modelos de IA en imágenes médicas, pero llevar estos modelos a producción sigue siendo un reto significativo.

Para realizar el proyecto se han utilizado GitHub Actions para automatizar y gestionar los servicios de AWS, permitiendo empaquetar los modelos y ejecutarlos sin necesidad de intervención humana. Además, se ha configurado el sistema para que realice la inferencia de manera autónoma cada vez que se sube una imagen a Amazon S3.

Esta automatización reduce el tiempo necesario para llevar un modelo desde la fase de desarrollo hasta la producción, garantizando consistencia y escalabilidad. El proyecto demuestra la eficacia y la viabilidad de esta prueba de concepto, permitiendo a los investigadores centrarse en la optimización de los modelos en lugar de en las tareas de despliegue, mejorando así la eficiencia y reduciendo los costos operativos.

Abstract.

This project came about in response to the need to facilitate the automated deployment of artificial intelligence models for medical image inference using Amazon Web Services (AWS). Artificial intelligence is transforming the field of medicine, improving its efficiency and accuracy in the diagnosis and treatment of diseases. In this context, MONAI (Medical Open Network for AI) provides an open source platform for the development of AI models in medical imaging, but bringing these models into production remains a significant challenge.

To implement the project, GitHub Actions has been used to automate and manage the AWS services, allowing the models to be packaged and executed without human intervention. In addition, the system has been configured to perform the inference autonomously each time an image is uploaded to Amazon S3.

This automation reduces the time required to take a model from development to production, ensuring consistency and scalability. The project demonstrates the effectiveness and viability of this proof of concept, allowing researchers to focus on model optimisation rather than deployment tasks, thus improving efficiency and reducing operational costs.

Índex

Índex de figures

Índex de codis

Abreviacions

AMI - Amazon Machine Image

AWS - Amazon Web Services

CLI - Command Line Interface

CI/CD - Continuous Integration/Continuous Deployment

DICOM - Digital Imaging and Communication In Medicine

EBS - Elastic Block Store

EC2 - Elastic Compute Cloud

ECR - Elastic Container Registry

IAM - Identity and Access Management

IA - Intel·ligència Artificial

MAP - MONAI Application Package

MONAI - Medical Open Network for AI

SSM - Simple Systems Manager

S3 - Simple Storage Service

SNS - Simple Notification Service

SQS - Simple Queue Service

1 Introducció

La intel·ligència artificial (IA) està transformant el camp de la medicina de maneres que fa pocs anys eren inimaginables. La seva capacitat per processar grans volums de dades amb precisió i velocitat està permetent avenços significatius en el diagnòstic i tractament de malalties. Les eines d'IA, com Mia[1], estan demostrant ser especialment valuoses en la detecció precoç de càncers i altres malalties, identificant signes que poden passar desapercebuts per als metges. Aquesta tecnologia no només augmenta la taxa de detecció, sinó que també redueix el temps d'espera per als resultats, millorant així la qualitat de l'atenció als pacients i augmentant les seves possibilitats de supervivència. A més, la IA pot ajudar a alleujar la càrrega de treball dels professionals de la salut, permetent-los dedicar més temps a la interacció directa amb els pacients i a la presa de decisions.

El creixent ús de la intel·ligència artificial (IA) en el sector sanitari ha donat lloc a la creació d'eines especialitzades per a l'anàlisi d'imatges mèdiques. Una de les plataformes més innovadores i influents en aquest àmbit és MONAI (Medical Open Network for AI). El projecte, desenvolupat inicialment per NVIDIA conjuntament amb el King's College de Londres, ofereix un marc de treball de codi obert basat en PyTorch, en el qual els investigadors poden accelerar la recerca i la col·laboració clínica en imatges mèdiques.

Llançada per primera vegada el 2019, MONAI ofereix una plataforma que és fàcil d'utilitzar, proporciona resultats reproduïbles i està optimitzada per a les necessitats específiques de les dades mèdiques, capaç de gestionar formats únics, resolucions i meta-informació especialitzada d'imatges mèdiques. Un dels seus objectius clau és permetre la reproductibilitat dels experiments, de manera que els investigadors puguin compartir i comparar resultats. Això és especialment important per poder validar i perfeccionar els models.

MONAI es divideix en tres components clau que ofereixen funcionalitats específiques per a diferents necessitats dels investigadors i professionals de la salut. Aquestes eines permeten cobrir tot el procés per al desenvolupament de models, des de la recerca fins a la producció clínica.

1. **MONAI Label** és una eina intel·ligent d'etiquetatge d'imatges que utilitza assistència d'IA per reduir el temps i l'esforç d'anotar nous conjunts de dades.
2. **MONAI Core** és la biblioteca principal del Projecte MONAI i proporciona eines i funcionalitats específiques per processar les dades i entrenar models d'IA per a imatges mèdiques.
3. **MONAI Deploy** té com a objectiu empaquetar, provar, desplegar i executar aplicacions d'IA en producció clínica.

[1] Mammography Intelligent Assessment és una plataforma d'IA per a la detecció de càncer de mama.

En els darrers anys, la intel·ligència artificial ha demostrat tenir un potencial transformador en diversos sectors. Tot i això, una gran part dels projectes d'IA no aconsegueixen arribar a la producció, quedant-se en fases inicials com a proves pilot. Segons la darrera enquesta realitzada per KDnuggets [1], un 80% dels models d'intel·ligència artificial mai arriben a la producció. Automatitzar aquest procés no només millora l'eficiència, sinó que també garanteix consistència, escalabilitat i seguretat, permetent que més models passin del laboratori a la pràctica real i aportin un valor tangible a les empreses i institucions.

Automatitzar el desplegament de les IA permet reduir el temps necessari per portar un model des de la fase de desenvolupament fins a la producció. Un desplegament manual pot ser lent i subjecte a errors humans, mentre que l'automatització assegura una transició més ràpida i eficient. A més, el desplegament al núvol és especialment important perquè permet l'accés a diversos serveis i recursos sense necessitat de mantenir maquinari físic. Això facilita l'escalabilitat, la flexibilitat i la capacitat d'executar models d'IA en entorns distribuïts, millorant així l'eficiència i reduint els costos operatius.

El núvol ofereix una infraestructura robusta i adaptable que permet gestionar grans volums de dades i processar-los amb alta velocitat. Això és crucial per als projectes d'IA, ja que sovint requereixen recursos computacionals intensius que no són fàcilment accessibles a través d'infraestructures locals. Utilitzant serveis al núvol, les organitzacions poden accedir a recursos sota demanda, pagant només pel que utilitzen, la qual cosa optimitza les despeses i evita inversions elevades en maquinari i en manteniment.

Amazon Web Services (AWS) és l'empresa líder en serveis de núvol, oferint una àmplia gamma de serveis que cobreixen des de l'emmagatzematge de dades fins al processament i l'anàlisi avançada. La seva infraestructura global i les seves solucions de seguretat robustes fan que AWS sigui una opció preferida per a empreses que busquen desplegar i gestionar models d'IA de manera eficient i segura. AWS proporciona eines específiques com Amazon SageMaker per entrenar models d'IA, permetent als desenvolupadors crear, entrenar i desplegar models. A més a més, AWS disposa de serveis com Amazon HealthLake i Amazon HealthImaging, dissenyats per gestionar i analitzar dades mèdiques, oferint solucions avançades per a la salut digital i millorant la capacitat de les institucions sanitàries per gestionar grans volums d'informació de manera eficient i segura.

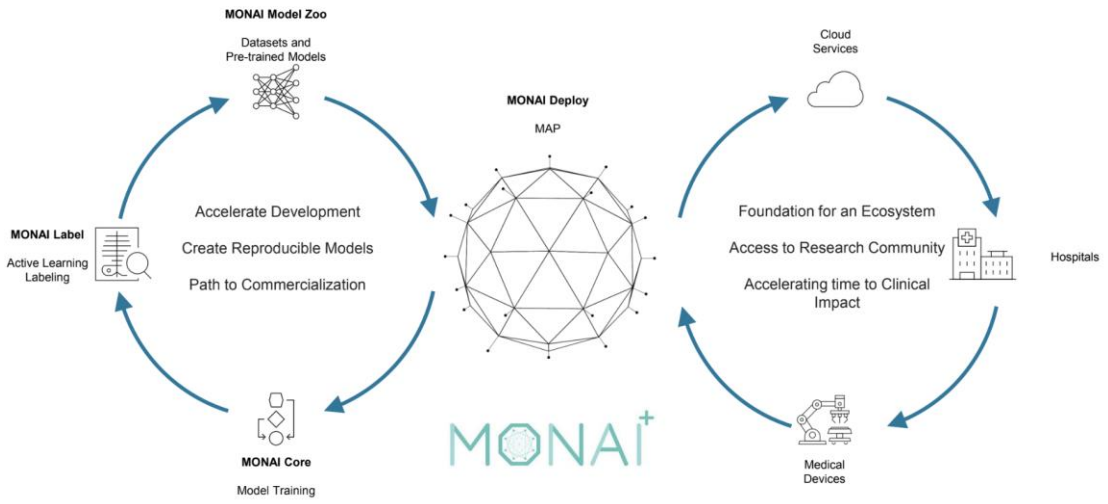


Figura 1 Esquema del cicle del desenvolupament de models amb MONAI

L'esquema il·lustra el cicle complet del desenvolupament de models d'IA mèdica amb MONAI, des de la fase inicial de desenvolupament del model fins al desplegament clínic. En el centre de l'esquema es troba el MONAI Application Package (MAP), que serveix com a punt de connexió clau entre aquestes dues fases. El que volem fer és que un cop tenim el model ja està entrenat, empaquetar-lo en un MAP per facilitar el seu desplegament en un entorn clínic. El MAP consisteix en una imatge d'un contenidor docker amb el model entrenat que conté una aplicació executable que donada una entrada, fa la inferència amb el model per generar les sortides.

Per aconseguir automatitzar el procés farem servir les GitHub Actions. GitHub Actions és una plataforma d'integració i desplegament continus (CI/CD) que permet automatitzar els processos de construcció, prova i desplegament de projectes. Amb GitHub Actions, es poden crear fluxos de treball que es desencadenen per esdeveniments específics, com ara quan algú puja codi al repositori (push) o quan crea una sol·licitud d'incorporació de canvis (pull request).

He triat GitHub Actions perquè GitHub és la plataforma de control de versions més popular i àmpliament utilitzada per desenvolupadors de tot el món, facilitant la gestió del codi font de manera col·laborativa i segura. Aquesta popularitat la fa una opció ideal per integrar automatització ja que, al tenir una base d'usuaris molt alta, existeix una gran quantitat de recursos, documentació i comunitats de suport. A més a més, permet executar els fluxes de treball (workflows) de manera gratuïta per als repositoris públics, la qual cosa resulta especialment beneficiosa per a projectes de codi obert.

1.1 Objectius

L'objectiu principal d'aquest projecte és fer una prova de concepte per automatitzar el procés de desplegament de models desenvolupats amb MONAI al núvol, facilitant que els investigadors puguin centrar-se en l'optimització dels models en lloc de les tasques de desplegament. Aquesta automatització ha de millorar l'eficiència del procés, reduint el temps i els costos associats al desplegament manual.

Aquest projecte està adreçat principalment a enginyers d'aprenentatge automàtic (machine learning), així com als enginyers de plataformes que es dediquen a crear infraestructures al núvol. L'automatització del desplegament a partir del codi font inclou la creació de l'estructura necessària al núvol, fent el procés més curt i simplificat, i abaratint els costos associats. Això permetrà que aquells professionals amb gran coneixement en aprenentatge automàtic, però sense experiència en tecnologies de núvol, puguin aprofitar les eines i serveis disponibles per a desplegar models automàticament, maximitzant així la seva productivitat.

En l'àmbit acadèmic, l'objectiu és expandir el coneixement i adquirir habilitats pràctiques que permetin afrontar els reptes tecnològics actuals. En l'actualitat, moltes empreses i institucions estan migrant a solucions basades en el núvol per aprofitar la seva flexibilitat, escalabilitat i eficiència en costos.

Durant la meva carrera, no he tingut l'oportunitat de treballar amb tecnologies de núvol, i per tant he volgut aprofitar aquesta ocasió per aprendre més sobre com funcionen. Això inclou entendre quins serveis ofereixen les principals plataformes de núvol, com ara AWS, el cost associat a la contractació d'aquests serveis, i com poden ser utilitzats per desplegar models d'intel·ligència artificial desenvolupats amb MONAI.

El coneixement adquirit en aquest procés no només em permetrà comprendre millor els avantatges i desafiaments de treballar amb el núvol, sinó que també em dotarà de les habilitats necessàries per aplicar aquestes tecnologies en projectes futurs. Així, estaré millor preparat per afrontar els reptes tecnològics de la meva carrera professional.

2 Planificació

Task Name	Duration	Start	Finish
Fase 1 Recerca i Aprenentatge	**16 days**	**Fri 01/03/24**	**Fri 22/03/24**
Creació del compte AWS	1 day	Fri 01/03/24	Fri 01/03/24
Investigació dels diferents serveis AWS	6 days	Fri 01/03/24	Sun 10/03/24
Proves inicials de serveis AWS	7 days	Mon 11/03/24	Tue 19/03/24
Sol·licitud d'augment de quota de CPU amb gràfica	1 day	Mon 11/03/24	Mon 11/03/24
Revisió de documentació i tutorials de Github Actions	3 days	Mon 18/03/24	Wed 20/03/24
Prova d'integració d'AWS amb GitHub Actions	4 days	Tue 19/03/24	Fri 22/03/24
Fase 2 Creació del MAP	**28 days**	**Sat 23/03/24**	**Fri 26/04/24**
Revisió documentació de MONAI i execució d'exemples	7 days	Sat 23/03/24	Fri 29/03/24
Disseny	2 days	Mon 01/04/24	Tue 02/04/24
Creació de l'AMI amb els paquets necessaris	3 days	Wed 03/04/24	Fri 05/04/24
Creació de la GitHub Action per a crear la Instància	5 days	Mon 08/04/24	Fri 12/04/24
Creació del flux de treball per a empaquetar el MAP	5 days	Sun 14/04/24	Thu 18/04/24
Joc de proves i correcció d'errors	6 days	Fri 19/04/24	Fri 26/04/24
Fase 3 Configuració de la Inferència	**18 days**	**Sat 27/04/24**	**Sun 19/05/24**
Disseny	4 days	Sat 27/04/24	Wed 01/05/24
Configurar notificacions S3	4 days	Mon 06/05/24	Thu 09/05/24
Enviar notificació a la instància EC2	4 days	Tue 07/05/24	Fri 10/05/24
Desenvolupament del script d'inferència	2 days	Sun 12/05/24	Mon 13/05/24
Joc de proves i correcció d'errors	6 days	Mon 13/05/24	Sun 19/05/24
Fase 4 Documentació	**50 days**	**Mon 01/04/24**	**Mon 03/06/24**
Introduccio	2 days	Sat 25/05/24	Sun 26/05/24
Desenvolupament projecte	40 days	Mon 01/04/24	Mon 20/05/24
Resultats	2 days	Sat 01/06/24	Sun 02/06/24
Conclusions	2 days	Sat 01/06/24	Sun 02/06/24
Abstract	1 day	Mon 03/06/24	Mon 03/06/24

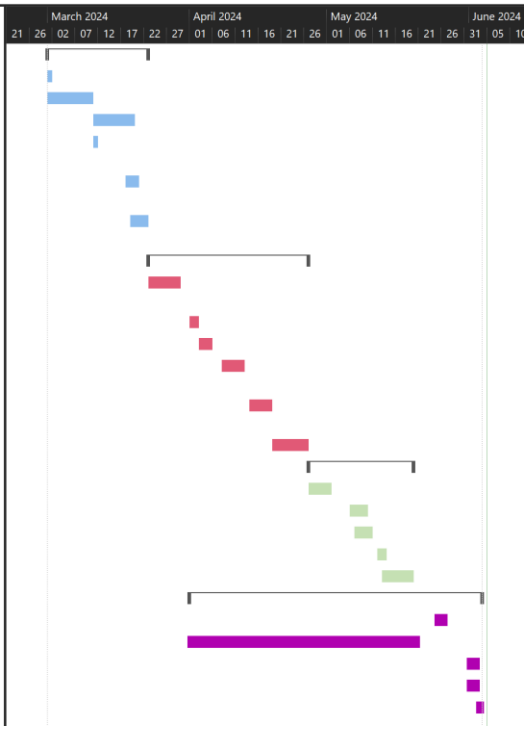


Figura 2 Diagrama de Gantt

Fase 1 Recerca i Aprenentatge

En aquesta primera fase, es va començar amb la creació del compte d'AWS, essencial per accedir als diversos serveis al núvol que ofereix la plataforma. Es va procedir a investigar i explorar els diferents serveis que proporciona AWS, amb l'objectiu d'entendre com podrien ser utilitzats per al projecte. Això va incloure la realització de proves inicials amb aquests serveis per familiaritzar-se amb el seu funcionament i les seves capacitats. A més, es va sol·licitar un augment de quota de CPU amb gràfica, necessari per treballar amb instàncies que incloguin GPU. Paral·lelament, es va revisar la documentació i els tutorials de GitHub Actions per comprendre com podrien ser integrats amb AWS per automatitzar els processos. Finalment, es van realitzar proves d'integració entre AWS i GitHub Actions per assegurar-se que les eines funcionaven correctament, utilitzant un usuari IAM i configurant la AWS CLI.

Fase 2 Creació del MAP

En aquesta etapa, es va començar amb una revisió de la documentació de MONAI i l'execució d'exemples per comprendre com funcionava. A continuació, es va procedir al disseny, identificant els paquets i configuracions necessàries per crear una imatge de màquina (AMI) amb tots els components requerits. Un cop identificats els requisits, es va crear l'AMI amb els paquets necessaris, incloent-hi els controladors de NVIDIA, Docker i altres eines essencials. Després, es va desenvolupar una GitHub Action per automatitzar la creació de la instància EC2, seguit de la creació d'un flux de treball que permetés empaquetar el MAP de manera eficient. Finalment, es va dur a terme un joc de proves exhaustiu per assegurar-se que tot funcionava correctament, corregint qualsevol error que es presentés durant el procés.

Fase 3 Configuració de la Inferència

En aquesta fase, es va començar amb el disseny de la solució, incloent la configuració de notificacions de S3 per assegurar que cada vegada que es puja una imatge, es generi una alerta automàtica. Es va configurar l'enviament d'aquestes notificacions a la instància EC2 per tal que aquesta pugui processar-les. A continuació, es va desenvolupar l'script d'inferència que descarrega les dades, executa el model i puja els resultats. Per acabar, es va dur a terme un joc de proves exhaustiu per verificar que tot funcionava correctament i es van corregir els errors identificats per garantir una operació fluida i precisa.

Fase 4 Documentació

Durant aquesta fase final, es va redactar tota la documentació necessària per al projecte. Això inclou una introducció detallada, el desenvolupament del projecte, els resultats obtinguts i les conclusions finals. La documentació assegura que qualsevol persona que vulgui replicar o entendre el projecte tingui tota la informació necessària de manera clara i organitzada.

3 Desenvolupament del projecte

3.1 Requisits

El sistema ha de permetre el desplegament automàtic del model d'aprenentatge automàtic i la configuració del procés d'inferència. Això implica la instal·lació i posada en marxa del model en un entorn de producció, així com l'establiment de les configuracions necessàries per a la seva operació continuada.

El sistema ha d'utilitzar els serveis d'Amazon Web Services (AWS) per al desplegament i execució del model de machine learning. Això inclou, però no es limita a, serveis com Amazon S3 per a l'emmagatzematge de dades, AWS Lambda per a la computació sense servidors, Amazon SageMaker per a la gestió del machine learning, i altres serveis rellevants d'AWS.

El sistema ha d'utilitzar GitHub com a plataforma per al control de versions i la col·laboració en el codi. Això permetrà la gestió centralitzada del codi font, la col·laboració entre els desenvolupadors, i la integració contínua i desplegament continu (CI/CD).

Restriccions:

- Només es desenvoluparà el backend, sense incloure components de frontend.
- El sistema ha de ser fàcil de configurar, pensant en l'ús per part d'un enginyer amb coneixements de machine learning però sense experiència prèvia en AWS.

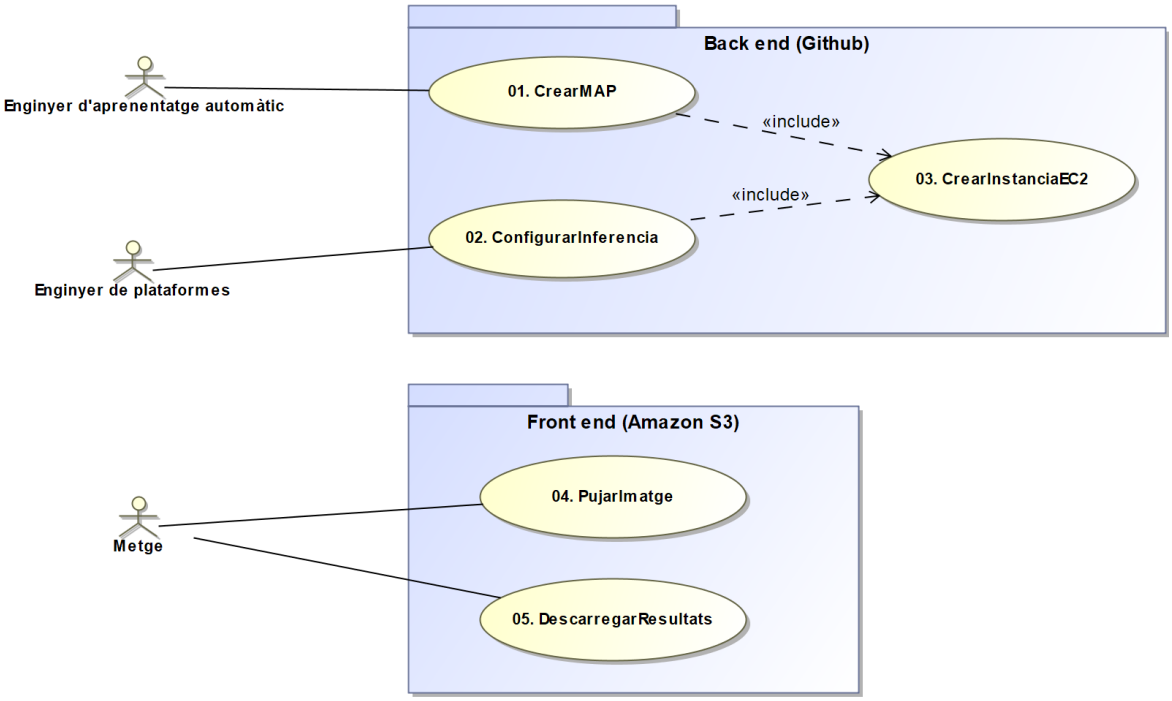


Figura 3 Diagrama casos d'ús

El projecte s’estructura en dos components principals: el backend i el frontend. El backend, gestionat principalment amb GitHub Actions, està centrat en la creació i configuració del MAP, així com en la instància EC2 que executa la inferència. Aquesta part del projecte ha estat el nostre focus principal, garantint que els processos d'automatització i desplegament funcionin de manera eficient i sense necessitat d'intervenció humana.

Per fer aquesta prova de concepte, s’utilitzarà directament Amazon S3. No obstant això, cal remarcar que en una versió definitiva del projecte, seria necessari implementar una interfície web que permetés als usuaris finals, com els metges, interactuar amb el sistema de manera més intuïtiva i amigable. Aquesta interfície web hauria de permetre la pujada d'imatges i la visualització o descàrrega dels resultats sense haver d'accedir directament a S3, millorant així l'experiència d'usuari i assegurant una major seguretat i control sobre les dades.

3.2 Disseny

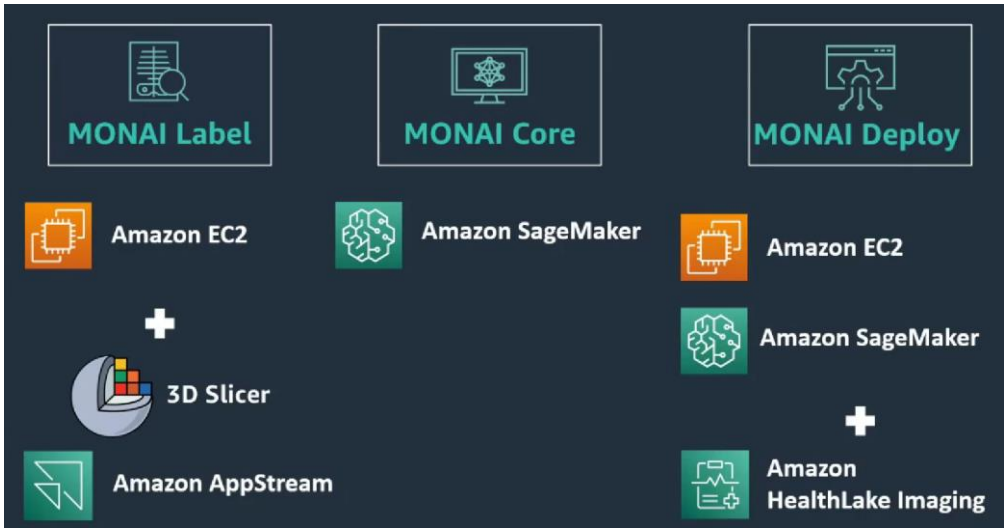


Figura 4 Serveis d'AWS on es pot executar MONAI [2]

En la part del desplegament, veiem que tenim dues alternatives, es pot executar tant en Elastic Compute Cloud (EC2) com en Amazon Sagemaker.

Amazon EC2 és un servei que proporciona capacitat de computació escalable al núvol. Està dissenyat per facilitar als desenvolupadors una infraestructura al núvol segura i escalable que redueix els costos de tenir maquinària física. Amazon EC2 permet als usuaris llogar màquines virtuals on poden executar les seves aplicacions, oferint la flexibilitat de configurar-les segons les necessitats específiques de cada projecte, ja que a l'hora de crear una instància, es pot configurar perquè la màquina virtual s'executi en diversos entorns o amb característiques específiques.

Amazon SageMaker és un servei que reuneix diverses eines que permeten als desenvolupadors i científics de dades crear, entrenar i desplegar models d'aprenentatge automàtic (machine learning). SageMaker simplifica cada pas del procés, des de la preparació de dades fins a l'entrenament i la inferència, proporcionant eines integrades que faciliten el desenvolupament i desplegament de models d'IA. A més, facilita el desplegament de models entrenats en entorns de producció, oferint inferència en temps real i permetent als usuaris escalar els seus serveis d'IA segons les necessitats.

En un principi ens vam plantejar fer el projecte en sagemaker, però ens vam adonar que ens aniria més bé fer-ho en EC2. Hi ha molts exemples de com desplegar els models en Sagemaker, i tots fan servir jumpyter notebooks. Els Jupyter Notebooks són una eina de codi obert que permet als usuaris crear i compartir documents que contenen codi executable, visualitzacions i text explicatiu tot en un mateix fitxer.

Tal com hem comentat en la introducció, el que volem fer és automatitzar el desplegament usant com a intermediari un MAP (vegeu figura 1) i, per tant, no ens serveix el sagemaker perquè en comptes d'usar el MAP, usa els models directament per a fer la inferència.

3.2.1 Creació del MAP

El primer pas per passar els models a producció és empaquetar l'aplicació juntament amb el model en un MAP, aquesta imatge de Docker compleix amb una sèrie d'especificacions que asseguren que l'aplicació continguda sigui fàcilment portable assegura que es pot executar en diferents entorns sense necessitat de modificacions.

Per crear el MAP és necessari disposar d'una targeta gràfica compatible amb NVIDIA CUDA. Atès que el portàtil des d'on treballo no és compatible, i molts altres programadors probablement tampoc disposen d'aquest recurs, he decidit automatitzar la creació del MAP mitjançant la reserva d'una instància amb GPU a EC2. Aquesta solució permetrà als desenvolupadors crear MAPs de manera eficient i sense les limitacions de maquinària.

Per reservar una instància a EC2 es pot fer mitjançant comandes de la línia de comandaments (CLI) d'AWS. No obstant això, he decidit utilitzar Terraform. L'ús de Terraform permet fer configuracions de manera més senzilla i reutilitzable, ja que només cal modificar el fitxer *main.tf* per ajustar les configuracions necessàries. El més important d'aquest fitxer de configuració és el tipus d'instància i la Imatge de Màquina de Amazon (AMI).

Com que necessitem una instància amb gràfica hem de triar entre instàncies:

- P3 tenen fins 8 GPU NVIDIA Tesla V100.
- P4 tenen fins 8 GPU NVIDIA Tesla A100.
- G3 tenen fins 4 GPU NVIDIA Tesla M60.
- G4 tenen fins 4 GPU NVIDIA T4.
- G5 tenen fins 8 GPU NVIDIA A10G.

Les AMI són imatges preconfigurades que proporcionen la informació necessària per llançar una instància EC2. Una AMI inclou el sistema operatiu, el servidor d'aplicacions i les aplicacions necessàries per al funcionament de la instància. Les AMI permeten als usuaris crear còpies exactes de les seves configuracions i desplegar múltiples instàncies amb la mateixa configuració en qüestió de minuts. Això facilita la gestió i l'escalabilitat dels serveis al núvol, ja que els usuaris poden seleccionar una AMI específica per adaptar-se a les seves necessitats de càlcul i programari. A més, les AMI poden ser personalitzades per incloure aplicacions específiques, controladors, configuracions de seguretat.

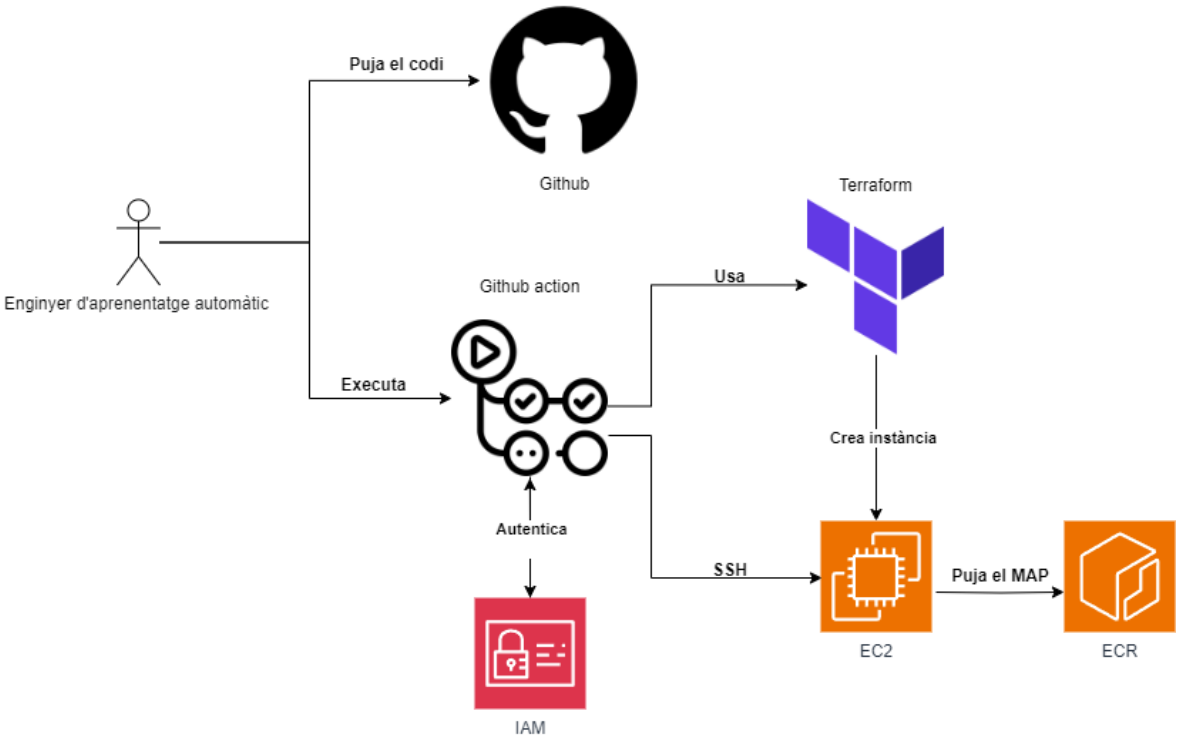


Figura 5 Diagrama Creació del MAP

El diagrama il·lustra el procés de creació del MAP. El procés comença quan l'enginyer d'aprenentatge automàtic puja el codi al repositori de GitHub. Tot seguit, s'executa el flux de treball que primer s'autentica a AWS mitjançant credencials IAM per tenir accés als serveis.

Un cop autenticat, el flux de treball utilitza Terraform per gestionar i crear una instància EC2. Aquesta màquina virtual està configurada amb els paquets i eines requerits per empaquetar el model.

En acabat, la GitHub Action es connecta a la instància EC2 via SSH per crear i pujar el MAP a un registre de contenidors ECR (Elastic Container Registry).

3.2.2 Configuració de la inferència

En aquesta part, el que volem fer és crear una nova instància que faci automàticament la inferència cada cop que algú pengi una nova imatge o arxiu ZIP en els contenidors de S3. Això implica configurar un sistema on la instància EC2 processa les dades i genera resultats de manera autònoma. Els resultats obtinguts de la inferència es pujaran al mateix contenidor de S3 per a ser descarregats pels usuaris, garantint un accés ràpid i fàcil als resultats. A més a més, s'importaran a HealthImaging, on es podran analitzar les metadades de les imatges Digital Imaging and Communication In Medicine (DICOM) i visualitzar-les posteriorment. Aquest enfocament permet que la inferència es realitzi de manera contínua i eficient, assegurant que les noves dades es processen immediatament després de ser penjades, proporcionant respostes ràpides als usuaris.

En la primera part, creo una instància per fer el MAP i ara, creo una altra instància per executar el MAP i fer la inferència. La justificació de fer-ho en dues màquines diferents és la següent. Un cop configurada la segona màquina, aquesta simplement executa l'última imatge pujada al registre ECR. Això significa que podem actualitzar el model i pujar-lo a l'ECR tantes vegades com sigui necessari sense haver de reconfigurar la màquina d'inferència. Aquesta separació de tasques fa que el procés sigui més modular i manté la màquina d'inferència transparent respecte a les actualitzacions del model. Així, qualsevol canvi o millora en el model es pot desplegar ràpidament i de manera eficient, sense interrompre el flux de treball de la inferència.

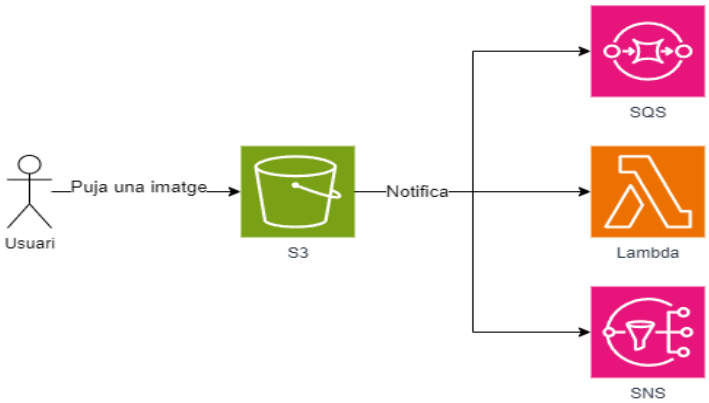


Figura 6 Esquema notificacions d'esdeveniments d'Amazon S3

Amb la màquina ja creada, hem de fer que d'alguna manera cada vegada que es pugi una imatge al contenidor de S3, es descarregui el fitxer i executi el model. En S3, es pot configurar perquè enviï notificacions quan algú crea un objecte de tres formes diferents:

- **Funció Lambda**: Les funcions Lambda són fragments de codi que s'executen en resposta a esdeveniments i permeten l'execució automàtica de tasques sense necessitat de gestionar servidors. Això significa que, cada vegada que es puja una imatge a un contenidor S3, es pot activar automàticament una funció Lambda que s'encarregui de descarregar el fitxer pujat i d'executar el model. Utilitzar Lambda per aquest propòsit proporciona una solució escalable i eficient, ja que les funcions Lambda només s'executen quan es produeixen els esdeveniments especificats, reduint la necessitat de recursos permanents i permetent una resposta ràpida i automatitzada a les pujades de fitxers.

- **Servei Simple de Notificació (SNS):** És un servei de missatgeria basat en el model productor/subscriptor, on els missatges són publicats en un tema (topic) i tots els subscriptors d'aquest reben les notificacions. Els subscriptors poden ser altres serveis, aplicacions o persones que necessiten rebre l'avís. Aquest enfocament facilita l'enviament de missatges a múltiples destinataris simultàniament, assegurant que tots els serveis necessaris rebin la notificació permeten una comunicació asíncrona i escalable.

- **Servei de Cua Simple (SQS)**: Les notificacions es poden enviar a una cua SQS, que seran llegides per un servidor. Amazon SQS és un servei de missatgeria completament gestionat que permet desconnectar i escalar components distribuïts, sistemes i aplicacions. En enviar les notificacions a una cua SQS, s'assegura que cada missatge (com ara la pujada d'una nova imatge) es processi de manera seqüencial. Un servidor pot llegir els missatges de la cua, descarregar les imatges pujades i executar el model corresponent. Aquest enfocament permet gestionar el processament de les imatges de manera ordenada i fiable, assegurant que no es perdin missatges i que es processin en l'ordre en què han arribat.

Al final m'he decidit per la primera opció. Per rebre els missatges SNS, has de deixar la màquina contínuament escoltant el port, cosa que pot resultar costós i ineficient. En les cues, els missatges s'emmagatzemen fins que un servidor els llegeix, cosa que pot requerir mantenir el servidor actiu tot el temps o manualment anar parant i engegant la màquina. En canvi, usant una funció Lambda podem engegar i aturar la instància només quan sigui necessari, reduint així els costos de tenir la màquina sempre activa. Aquesta solució és més eficient econòmicament, ja que les funcions Lambda només es facturen pel temps real d'execució i permeten una resposta immediata a les pujades de fitxers.

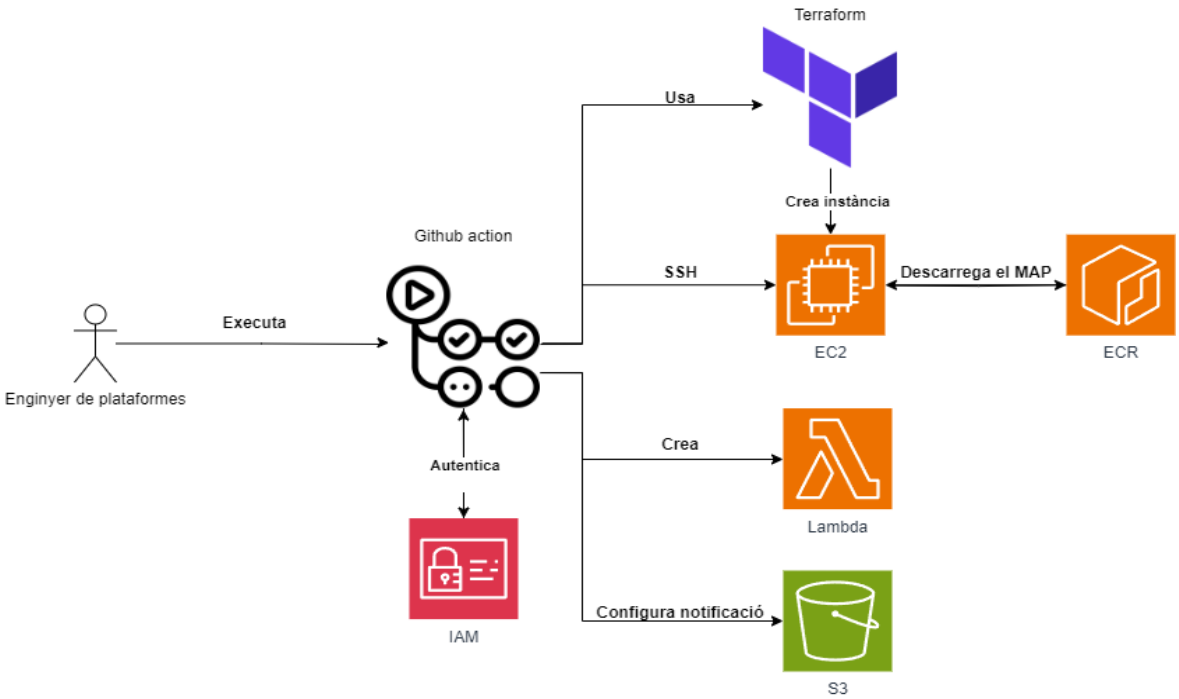


Figura 7 Diagrama configuració de la inferència

El diagrama mostra el procés de configuració de la inferència. Tot comença quan l'enginyer de plataformes executa el flux de treball. El primer pas de crear la instància és molt semblant a la creació del MAP, però en aquest cas, en comptes de pujar la imatge, la descarreguem des del ECR. A continuació, es crea una funció Lambda que serà l'encarregada d'iniciar la inferència. Finalment, es configura S3 perquè notifiqui a la funció Lambda cada vegada que es puja una nova imatge. Aquesta configuració permet que la funció Lambda activi la instància EC2 per descarregar la imatge del S3 i executar el procés d'inferència automàticament, assegurant una resposta ràpida a les pujades de fitxers.

3.2.3 Execució de la inferència

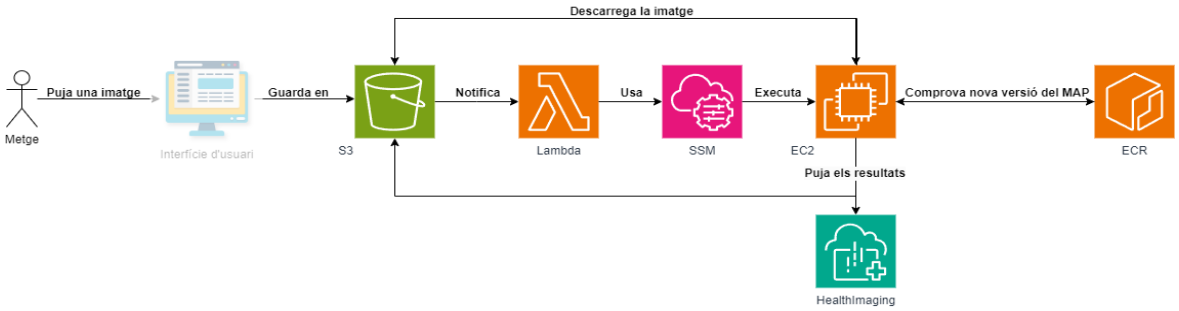


Figura 8 Diagrama execució de la inferència

El diagrama mostra el procés d'execució de la inferència. El procés comença quan el metge puja una imatge. Idealment, aquest pas es realitzaria a través d'una interfície d'usuari que facilitaria la interacció, però per a les proves, les imatges es pujaran directament a S3. Un cop la imatge està guardada en S3, el servei genera una notificació i engega una funció Lambda.

La funció Lambda utilitza el servei AWS Systems Manager (SSM) per iniciar la instància EC2 i executar l'script d'inferència. La instància EC2 descarrega la imatge des del S3 i comprova si hi ha una nova versió del MAP a ECR. Un cop completada la inferència, els resultats es pugen a HealthImaging per a la seva posterior anàlisi.

3.3 Implementació

3.3.1 Creació del MAP

Tal com recomanen des de MONAI, he utilitzat la instància g4dn.xlarge, que disposa d'una targeta gràfica NVIDIA T4 de 16 GB juntament amb 4 vCPU i 16 GB de RAM, i que té un cost de 0,526 dòlars per hora. Aquesta configuració proporciona la potència de càlcul suficient per poder empaquetar i executar els models. És molt important tenir en compte que, si és el primer cop que utilitzes màquines amb GPU en el teu compte d'AWS, has de sol·licitar un augment de la quota per obtenir permís per executar-les, ja que per defecte estan restringides.

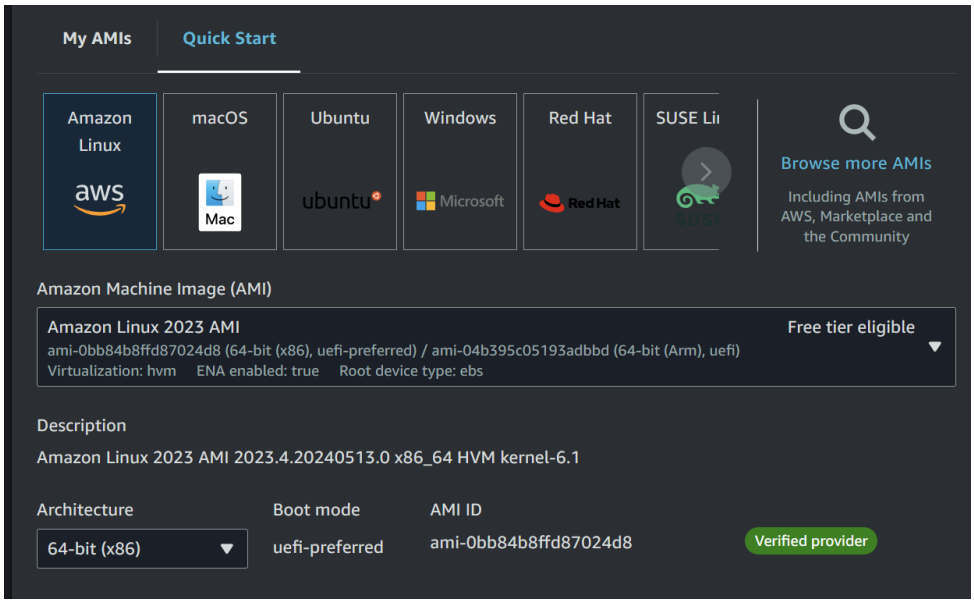


Figura 9 Interfície de selecció de les AMI a la consola de AWS

Inicialment, vaig crear una instància amb la imatge d'Ubuntu i vaig configurar tots els paquets necessaris per generar els MAP. La imatge d'Ubuntu per defecte és una imatge molt lleugera que només conté els components bàsics per funcionar. Per tal de preparar l'entorn, vaig haver de seguir diversos passos detallats:

- Instal·lació dels controladors NVIDIA: Com que la instància necessita suport per a GPU, vaig instal·lar els controladors NVIDIA per assegurar-me que la targeta gràfica fos reconeguda i utilitzada correctament pel sistema.
- Instal·lació de Docker: Docker és essencial per a la creació i execució de contenidors.
- Instal·lació de l'AWS CLI: L'eina de línia de comandaments d'AWS és necessària per gestionar els serveis d'AWS des de la instància.
- Instal·lació d'Anaconda: Per crear un entorn de desenvolupament Python virtual.

- Configuració de l'entorn virtual de Python: Amb Anaconda instal·lat, vaig crear un entorn virtual dedicat. Dins d'aquest entorn, vaig instal·lar les llibreries requerides, com ara MONAI, PyTorch i altres dependències necessàries per empaquetar i executar els models.

Tots aquests passos els vaig incloure en el script *ami.sh* per automatitzar el procés de configuració de la instància. Per evitar haver de reinstal·lar i configurar tot cada vegada que es crea una nova instància, cosa que suposaria una gran pèrdua de temps, vaig crear una AMI a partir d'una instància ja configurada. EC2 permet crear AMI a partir d'instàncies existents, capturant així tot l'entorn de configuració i instal·lacions necessàries.

Tot i que l'ús d'una AMI personalitzada és una solució eficient per estalviar temps i assegurar la consistència en la configuració de les instàncies, presenta alguns problemes, entre els quals destaca la disponibilitat regional. Les AMI són un recurs regional, la qual cosa significa que només estan disponibles a la regió des d'on es van crear i compartir. Per fer que una AMI estigui disponible en una regió diferent, cal copiar l'AMI a la regió desitjada i, posteriorment, compartir-la. Aquest procés implica costos per emmagatzematge i transferència de dades.

A més, un altre dels problemes significatius és el manteniment i l'actualització de les AMI. Mantenir una AMI actualitzada amb les últimes revisions de seguretat i versions de paquets pot ser una tasca desafiant. Per solucionar això, es pot utilitzar AWS Systems Manager Automation. Aquesta eina permet aplicar revisions de forma automàtica a una AMI amb les versions més recents dels paquets especificats.

Figura 10 AMIs que venen configurades amb els divers de NVIDIA

Al catàleg d'AMI es poden trobar diverses imatges que ja vénen preparades amb els controladors de NVIDIA instal·lats juntament amb Docker. Com que són imatges oficials, Amazon s'encarrega de mantenir-les actualitzades i segures amb les actualitzacions més recents. No obstant això, un cop engegada aquesta imatge, encara hem d'acabar d'instal·lar les llibreries de Python necessàries per al nostre projecte.

Un cop aclarides les característiques de les màquines virtuals que reservem, hem de parlar del flux de treball que seguim per crear el MAP. Aquest és el procés detallat:

1. El primer pas per crear el MAP és autenticar-nos amb les credencials d'AWS. Això es pot fer utilitzant la AWS CLI, que ja està instal·lada per defecte a les màquines virtuals de GitHub, o cridant a l'acció *configure-aws-credentials*. En qualsevol dels dos casos, no s'han d'escriure les credencials directament en el codi, sinó que s'han de guardar en els secrets del repositori. Els secrets són variables que permeten emmagatzemar informació confidencial i que s'encripten abans d'arribar a GitHub.

   A més a més, els executors allotjats per GitHub executen el codi dins de màquines virtuals efímeres i aïllades. Això assegura que no hi ha manera de comprometre persistentment aquest entorn ni accedir a més informació d'altres usuaris, ja que un cop finalitza el procés, la màquina virtual es "neteja". Aquesta característica proporciona una capa addicional de seguretat, garantint que no es poden reutilitzar ni comprometre les dades o claus temporals utilitzades durant l'execució del codi.

2. El segon pas és generar una clau RSA temporal per poder connectar-se a la instància que crearem mitjançant SSH.

3. El tercer pas és executar Terraform per crear la instància, assegurant-se de passar la clau pública creada anteriorment.

4. Un cop creada la instància, li assignem un rol. Els rols en AWS són una manera segura de concedir permisos per accedir a altres serveis d'AWS sense necessitat d'utilitzar credencials d'usuari. Els rols es defineixen a través del servei AWS Identity and Access Management (IAM) i s'assignen a les instàncies per permetre que actuïn amb els permisos especificats en el rol.

5. Amb la instància en funcionament, el següent pas és clonar el repositori de Git juntament amb el model necessari. Això ens permet tenir tot el codi de la nostra aplicació disponibles a la instància per poder empaquetar i crear el MAP.

6. Finalment, pugem la imatge Docker l'Amazon Elastic Container Registry (ECR). Amazon ECR és un servei de registre de contenidors completament gestionat que facilita l'emmagatzematge, la gestió i el desplegament d'imatges Docker.

Un cop acabat el flux de treball, podem parar o eliminar la instància, ja que no la tornarem a utilitzar. Quan es para una instància, aquesta deixa de funcionar i no consumeix recursos de càlcul, però els recursos d'emmagatzematge, com els volums Elastic Block Store (EBS) associats, es mantenen i es continuen aplicant càrrecs; la instància es pot reiniciar en qualsevol moment mantenint el seu estat anterior, tot i que la IP pública pot canviar. En canvi, quan s'elimina una instància EC2, aquesta es destrueix completament i deixa de generar costos associats.

Com que la clau RSA s'ha guardat a la màquina virtual de GitHub, un cop finalitzat el procés, ja no tindrem accés a aquesta clau i únicament ens podrem connectar accedint a la consola principal d'AWS.

3.3.2 Configuració de la inferència

El flux de treball seguit per a configurar la inferència és el següent:

1. El primer pas és crear la instància EC2, similar al procés de creació del MAP. En aquest cas, en lloc de pujar la imatge, la instància descarrega la imatge des del registre ECR.

   Al crear la instància, ja li copio un script preconfigurat *ec2.sh* que es responsabilitza de descarregar les dades des del contenidor S3, executar el model per realitzar la inferència, i finalment, pujar els resultats generats de nou a S3 i a HealthImaging. Per tant, la funció Lambda tan sols ha d'executar aquest script.

   Els scripts de configuració que es troben al repositori tenen les variables buides. Això és així perquè la informació que contenen és confidencial. Quan s'executa el workflow, la màquina virtual de GitHub substitueix el valor d'aquestes variables de manera local utilitzant els secrets configurats al repositori. D'aquesta manera, es manté la seguretat i la confidencialitat de la informació sensible. El codi següent mostra com es realitza aquesta substitució:

```
- name: Update ec2.sh with secrets
  run: |
    cd setup
    sed -i
's|ACCOUNT_ID=""|ACCOUNT_ID="${{secrets.AWS_ACCOUNT_ID}}"|' ec2.sh
    sed -i
's|ROLE_ARN=""|ROLE_ARN="${{secrets.AWS_MEDICALIMAGING_ROLE}}"|'
ec2.sh
    sed -i
's|DATASTORE_ID=""|DATASTORE_ID="${{secrets.AWS_DATASTORE_ID}}"|'
ec2.sh
    sed -i 's|REGION=""|REGION="${{secrets.AWS_REGION }}"|' ec2.sh
    sed -i
's|DOCKER_IMAGE_TAG=""|DOCKER_IMAGE_TAG="${{env.REPOSITORY}}:${{
env.TAG }}"|' ec2.sh

- name: Send ec2.sh to EC2 instance
  run: |
    scp -i  $SSH_PRIVATE_KEY_PATH -o StrictHostKeyChecking=no
setup/ec2.sh $VM_NAME:/home/ubuntu/ec2.sh
    ssh -i $SSH_PRIVATE_KEY_PATH $VM_NAME "chmod +x
/home/ubuntu/ec2.sh"
```

Codi 1 Substitució de les variables i enviament del script a la instància EC2

2. Una vegada creada la instància, es crea una funció Lambda que serà l'encarregada d'iniciar la inferència automàticament. Aquesta funció Lambda s'encarrega de gestionar l'execució del model d'IA en la instància EC2 quan es detecta un nou fitxer a S3. Això ho fem utilitzant SSM, ja que aquest permet enviar comandes a la instància EC2 per executar scripts de manera remota.

```
# Zip the Lambda function code
zip -r function.zip lambda_function.py

# Create or update the Lambda function
aws lambda create-function --function-name $FUNCTION_NAME \
--zip-file fileb://function.zip --handler
lambda_function.lambda_handler \
--runtime python3.8 --role $ROLE_ARN --region $REGION \
|| \
aws lambda update-function-code --function-name $FUNCTION_NAME \
--zip-file fileb://function.zip --region $REGION

# Add S3 bucket permissions to invoke the Lambda function
aws lambda add-permission --function-name $FUNCTION_NAME --
statement-id S3Invoke --action lambda:InvokeFunction --principal
s3.amazonaws.com --source-arn arn:aws:s3:::$BUCKET_NAME
```

Codi 2 Creació de la funció lambda

La funció en Python està en el repositori de GitHub, però es pot pujar la funció comprimida en un fitxer ZIP, tal com es mostra en el codi anterior. Aquesta pràctica facilita la gestió i actualització de les funcions Lambda, permetent desplegar versions noves simplement actualitzant el fitxer ZIP i fent una crida a l'API d'AWS per crear o actualitzar la funció Lambda.

3. Es configura el servei S3 perquè enviï notificacions a la funció Lambda cada vegada que es puja una nova imatge.

   El fragment de codi de la següent pàgina configura una notificació d'esdeveniments en un bucket de S3 per invocar una funció Lambda quan es crea un nou objecte. Primer, s'atorguen permisos a la funció Lambda perquè pugui ser invocada per S3. A continuació, es crea una configuració d'esdeveniments en format JSON, especificant que la funció Lambda serà invocada quan es produeixin esdeveniments de creació d'objectes al bucket, amb un filtre opcional per només considerar objectes amb el prefix "input/". Finalment, s'aplica aquesta configuració al bucket de S3 mitjançant l'API de S3, assegurant que qualsevol nou objecte carregat que compleixi els criteris definits desencadenarà l'execució de la funció Lambda.

```
# Add S3 bucket permissions to invoke the Lambda function
aws lambda add-permission --function-name $FUNCTION_NAME --statement-
id S3Invoke --action lambda:InvokeFunction --principal
s3.amazonaws.com --source-arn arn:aws:s3:::$BUCKET_NAME

# Create S3 event configuration JSON
cat <<EOT > s3_event_configuration.json
{
  "LambdaFunctionConfigurations": [
    {
      "LambdaFunctionArn":
"arn:aws:lambda:$REGION:$ACCOUNT_ID:function:$FUNCTION_NAME",
      "Events": ["s3:ObjectCreated:*"],
      "Filter": {
        "Key": {
          "FilterRules": [
            {
              "Name": "prefix",
              "Value": "input/"
            }
          ]
        }
      }
    }
  ]
}
EOT

# Apply the S3 event configuration
aws s3api put-bucket-notification-configuration --bucket $BUCKET_NAME
--notification-configuration file://s3_event_configuration.json
```

Codi 3 Configuració de la notificació

4 Resultats

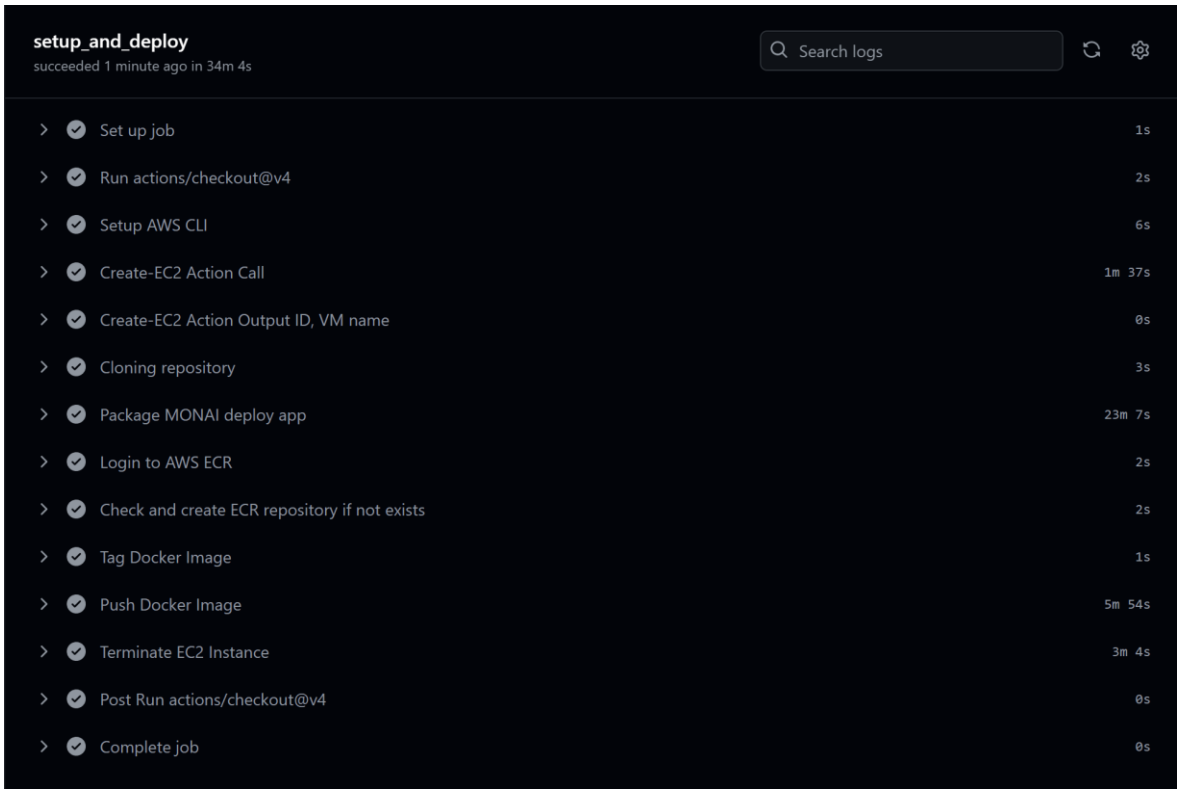


Figura 11 Resultats d'executar create_map.yml

Amb aquests resultats, podem concloure que és correcte eliminar les instàncies un cop finalitzat el procés, ja que el temps que triga a crear i iniciar la màquina és negligible en comparació amb el temps que tarda a construir i pujar el MAP. Com que només tinc quota de CPU per posar en marxa una única instància, abans de crear-ne una de nova, he d'eliminar o parar qualsevol altra instància que estigui activa.

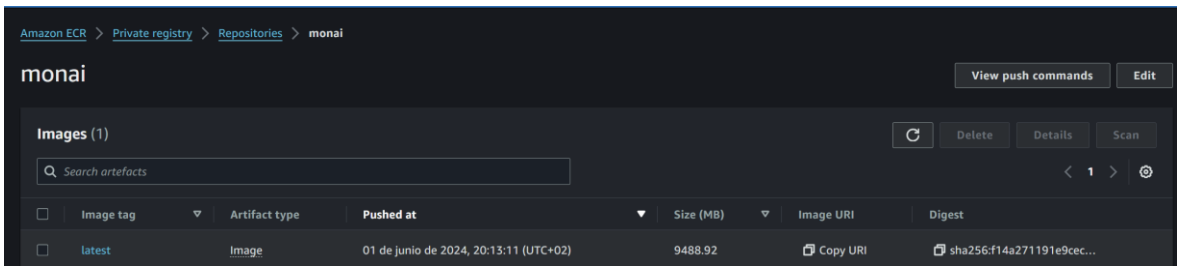


Figura 12 Imatge pujada a Amazon ECR

Tot aquest flux de treball ha sigut per a crear la imatge i pujar-la al repositori de Amazon ECR. Es pot observar que la imatge comprimida amb el model i tots lo necessari per funcionar ocupa unes 9 GB.

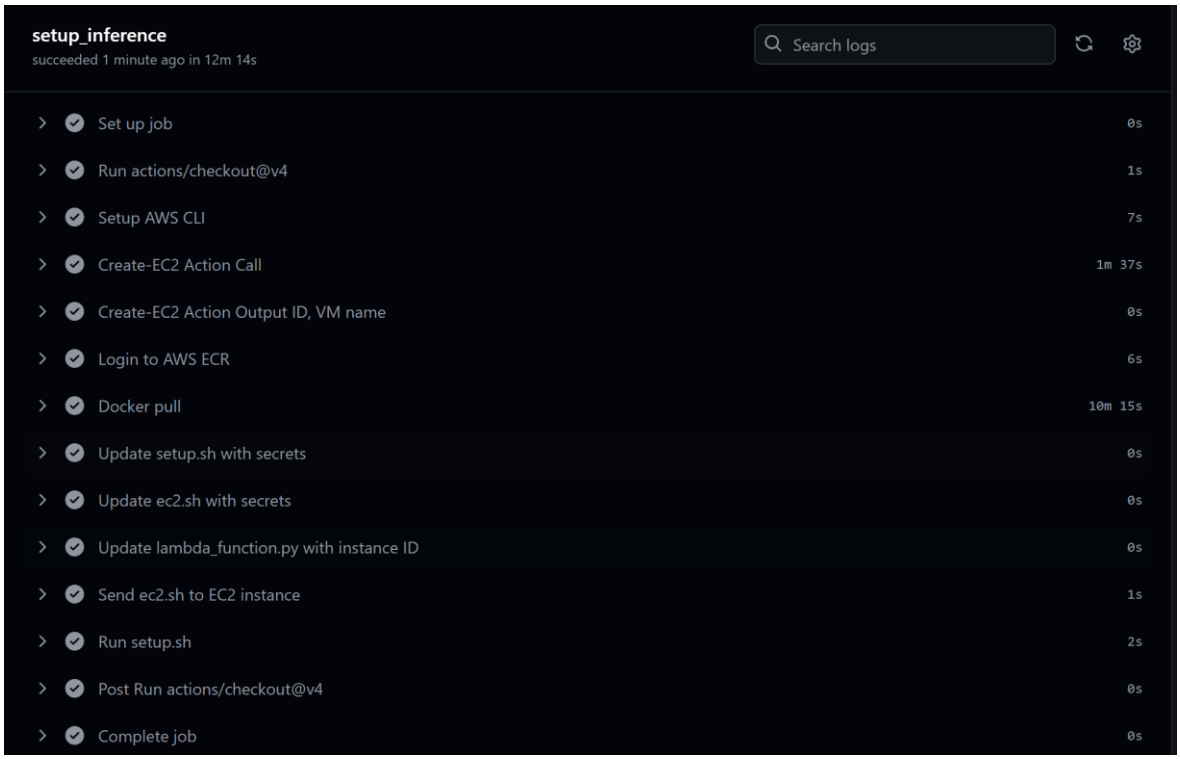


Figura 13 Resultat d'executar configure_inference.yml

Configurar la inferència és un procés bastant ràpid. Un cop engegada la màquina, s'actualitza els fitxers (localment dins la màquina virtual de github) amb les variables secretes i s'executa setup.sh per crear la funció lambda i connectar-la amb S3.

El pas que consumeix més temps és el *docker pull*, que descarrega la imatge Docker per tenir-la preparada. Encara que aquesta configuració inicial descarregui la imatge actual, si més endavant s'actualitza el MAP, la màquina EC2 tornarà a fer un *docker pull* per obtenir la versió més recent, assegurant que sempre s'utilitzi l'última versió del model.

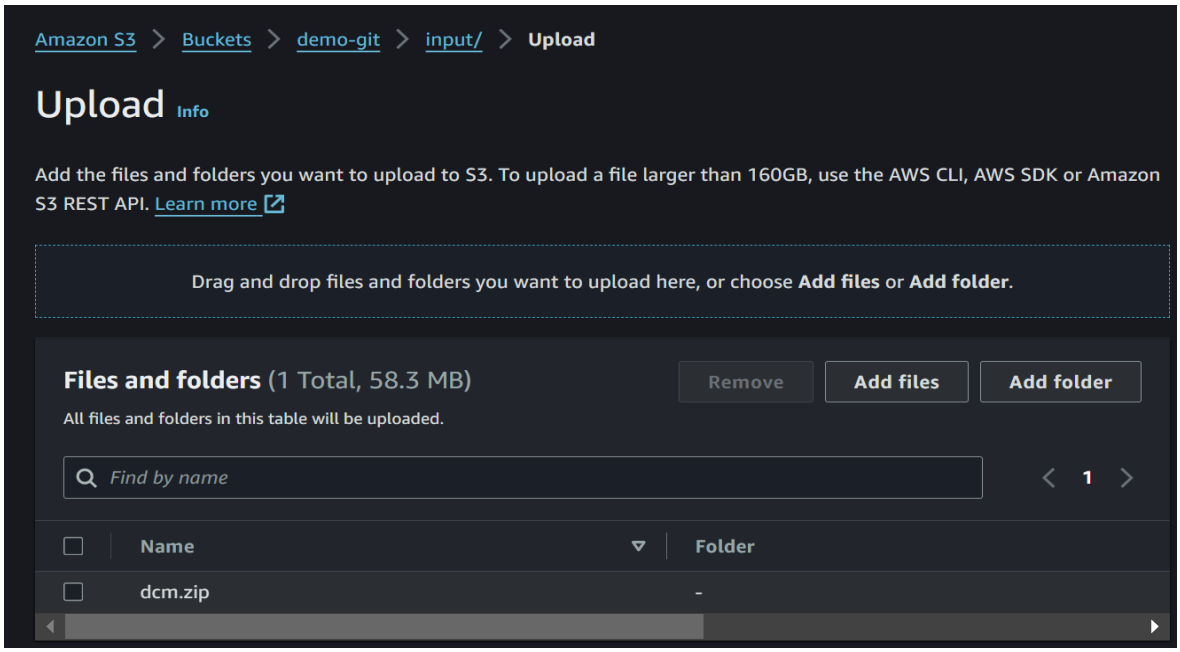


Figura 14 Pujar fitxers de prova

S'han pujat els fitxers de prova directament a S3, tal com es mostra a la imatge. En un escenari real, seria necessari desenvolupar un frontend que permetés als usuaris interactuar de manera més fàcil, intuïtiva i segura, sense que els usuaris tinguessin control directe sobre S3.

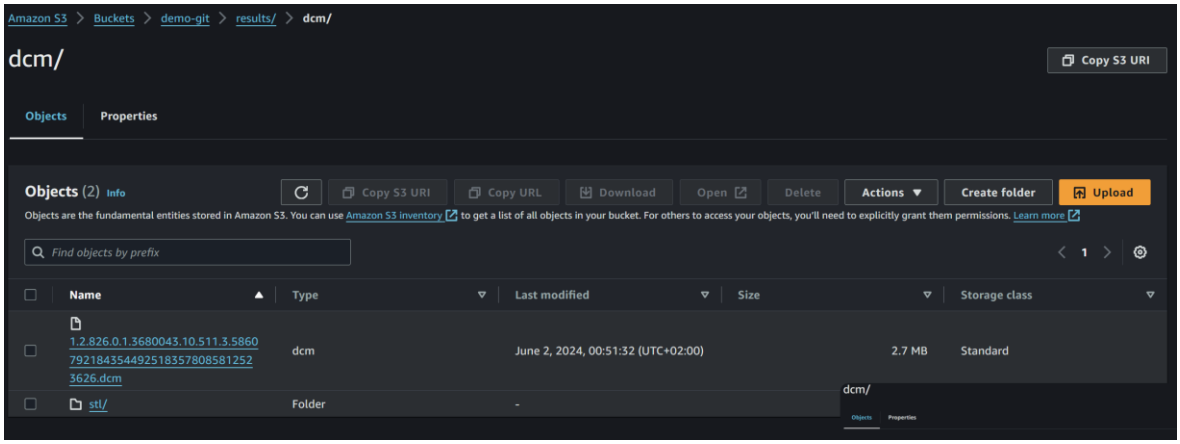


Figura 15 Resultats guardats en S3

Un cop pengem un fitxer de prova (dcm.zip) a la carpeta input, veiem que ens ha guardat els resultats en una carpeta amb el mateix nom. En aquest model que hem executat el que ens genera és una imatge DICOM junt amb un fitxer .stl amb la segmentació 3D.

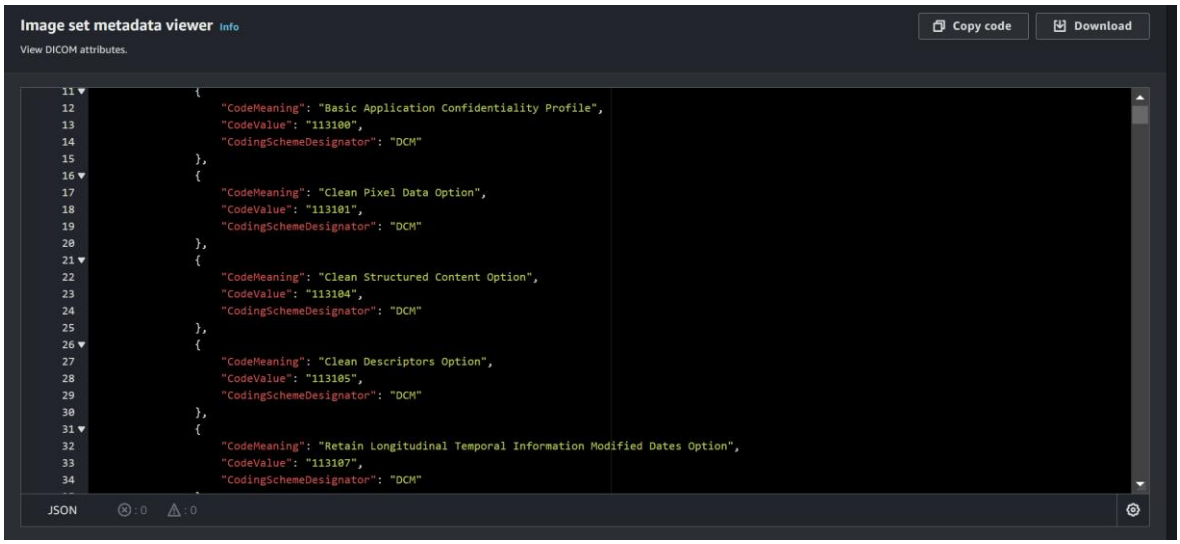


Figura 16 Resultats en HealthImaging

Les imatges també estan pujades al HealthLake, però només podem llegir les metadades dels DICOM. HealthImaging no permet visualitzar les imatges directament, no obstant això, es podria crear una interfície gràfica usant el codi que podem trobar a: https://github.com/aws-samples/aws-healthimaging-samples/tree/main/imaging-viewer-ui

Aquesta interfície permetria als usuaris interactuar de manera més efectiva amb les imatges mèdiques, millorant la seva experiència i facilitant l'anàlisi i la interpretació de les dades visuals.

per fer-ho més fàcil, podem connectar el HealthImaging amb FlexView, una eina que permet visualitzar i gestionar imatges mèdiques. FlexView ofereix la possibilitat de veure fins a 50 estudis únics de manera gratuïta. Un cop oberts, aquests estudis es poden visualitzar tantes vegades com es vulgui per qualsevol usuari del compte. Això permet visualitzar els DICOM sense necessitat de descarregar-los ni tenir cap programa específic instal·lat al PC.

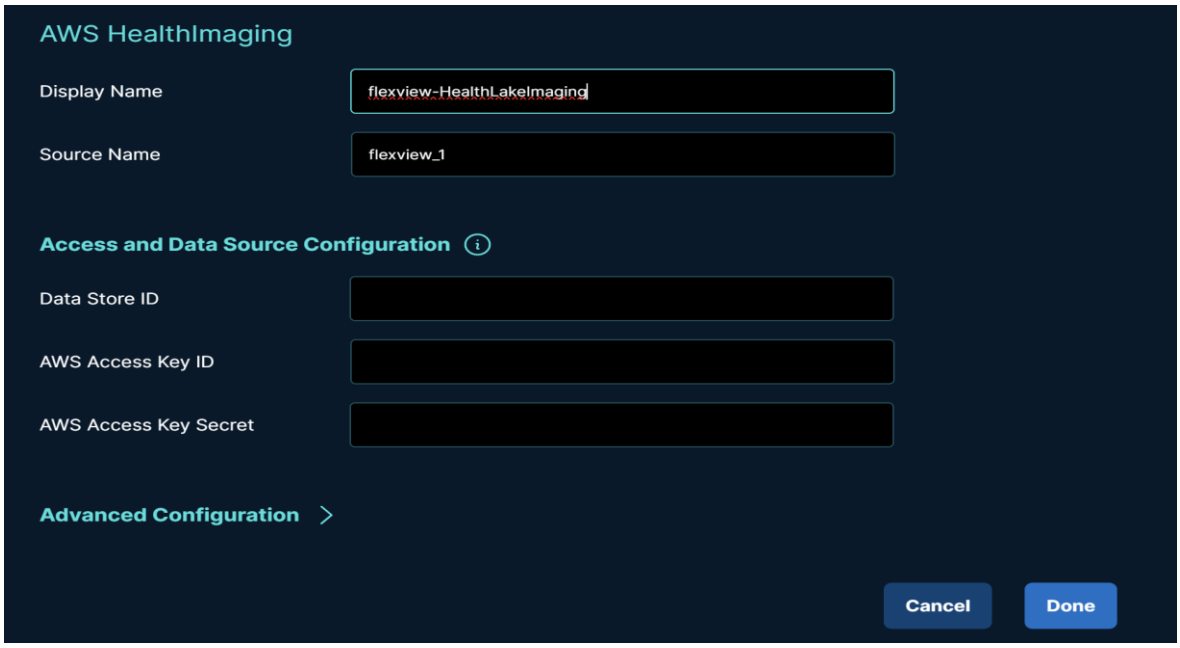


Figura 17 Configuració FlexView

Per connectar FlexView amb HealthImaging, només cal proporcionar l'ID del Data Store i les credencials d'un usuari IAM amb els permisos de lectura de HealthImaging per permetre la visualització de les imatges DICOM.

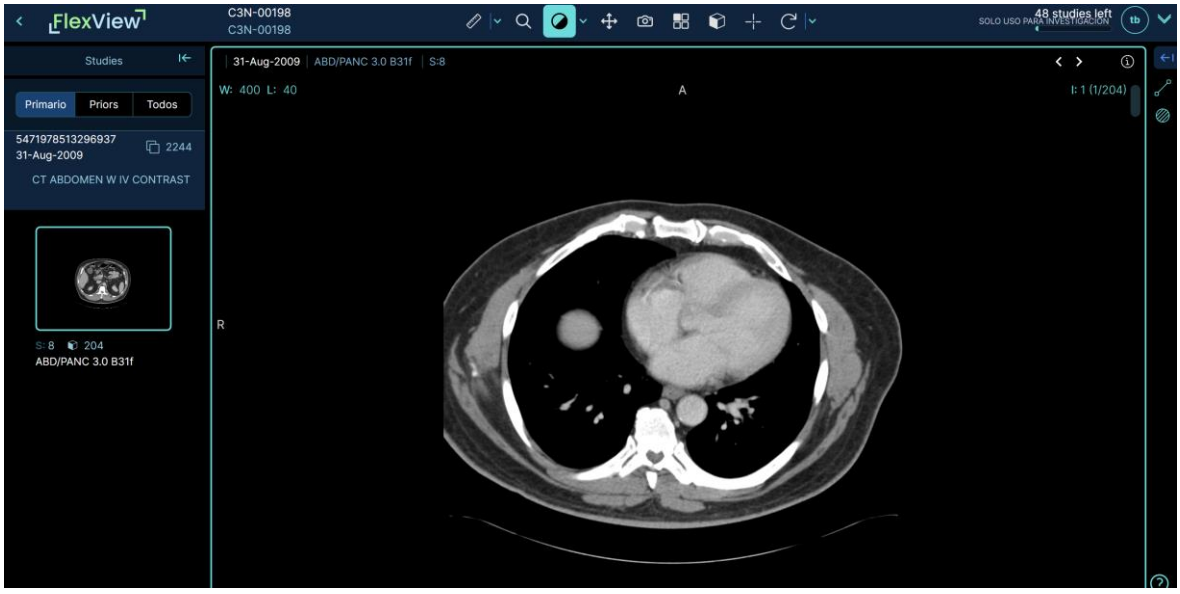


Figura 18 Visualització d'un DICOM en FlexView

5 Evaluació de costos

El cost de realització del projecte s'ha basat principalment en les hores que hi he dedicat i el cost dels serveis d'AWS, ja que GitHub i altres eines que he utilitzat són gratuïtes.

El temps total dedicat a la realització del projecte és d'aproximadament 300 hores. Gran part d'aquest temps ha estat invertit a aprendre a utilitzar tres tecnologies amb les quals no havia treballat abans: AWS, GitHub Actions i MONAI. Inicialment, vaig dedicar diverses hores a explorar la documentació i els tutorials de cadascuna d'aquestes eines per comprendre com funcionen i quines eren les millors pràctiques per a la seva implementació. A més, es van fer diverses proves per comprendre el funcionament dels serveis i la seva integració. Per exemple, es va provar la connexió de GitHub amb S3 per automatitzar la pujada i gestió d'arxius. Tot i que aquesta connexió no es va utilitzar en el codi final del projecte.

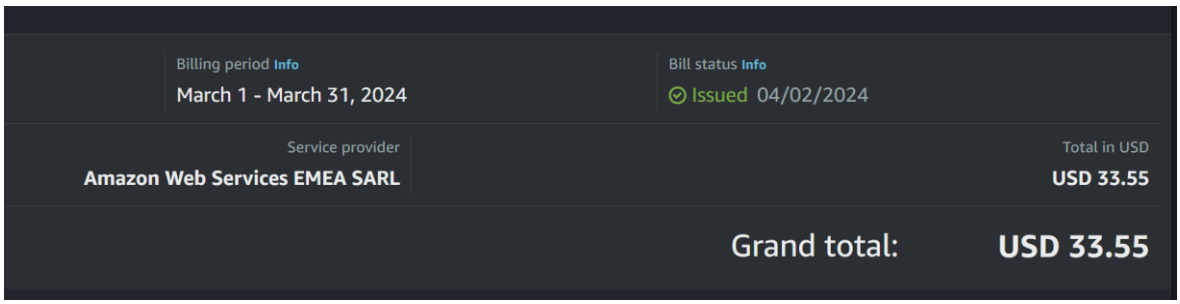
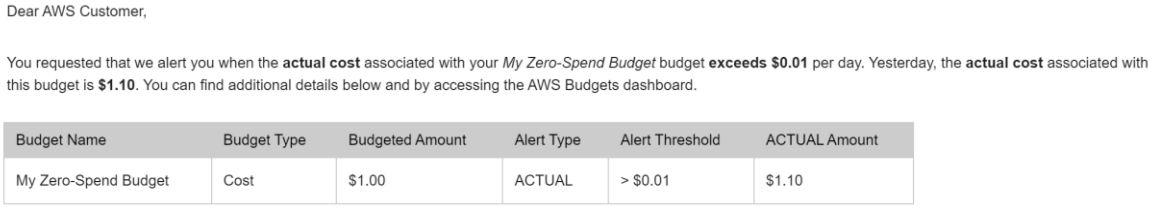


Figura 19 Factura AWS febrer

La factura del mes de febrer va ser deguda a un descuit. No sabia ben bé com funcionava el sistema i em vaig deixar un notebook de SageMaker actiu a la regió de Londres (eu-west-2). Quan entrava a SageMaker a la regió Virginia (us-east-1) no veia cap instància activa i, per tant, no vaig desactivar la instància de Londres fins al cap d'uns quants dies. A partir d'aquesta lliçó, vaig configurar una alarma que cada dia m'envia un correu electrònic si el dia anterior vaig gastar més de 0,01 dòlars, això m'ha permès evitar altres sorpreses i tenir un millor control sobre les despeses.



Dear AWS Customer,

You requested that we alert you when the **actual cost** associated with your *My Zero-Spend Budget* budget **exceeds $0.01** per day. Yesterday, the **actual cost** associated with this budget is **$1.10**. You can find additional details below and by accessing the AWS Budgets dashboard.

Budget Name	Budget Type	Budgeted Amount	Alert Type	Alert Threshold	ACTUAL Amount
My Zero-Spend Budget	Cost	$1.00	ACTUAL	> $0.01	$1.10

Figura 20 Exemple notificació quan s'activa l'alarma

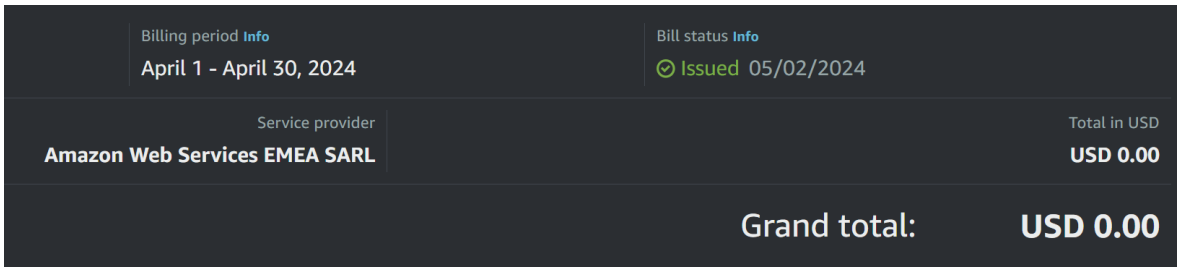


Figura 21 Factura AWS abril

A l'abril, tot i haver consumit recursos fora de la capa gratuïta d'AWS, la factura total va resultar ser de 0 dòlars gràcies al meu tutor, que em va ajudar a aconseguir un cupó de 100 euros per a realitzar el projecte. Aquest suport va ser essencial per poder utilitzar els serveis necessaris sense generar costos addicionals, permetent-me centrar en el desenvolupament i execució del projecte sense preocupacions. El cupó em va ser atorgat per participar en el programa Skills2Jobs, que premia els millors treballs finals de grau i màsters entre d'altres. Per mes informació accediu al següent enllaç.

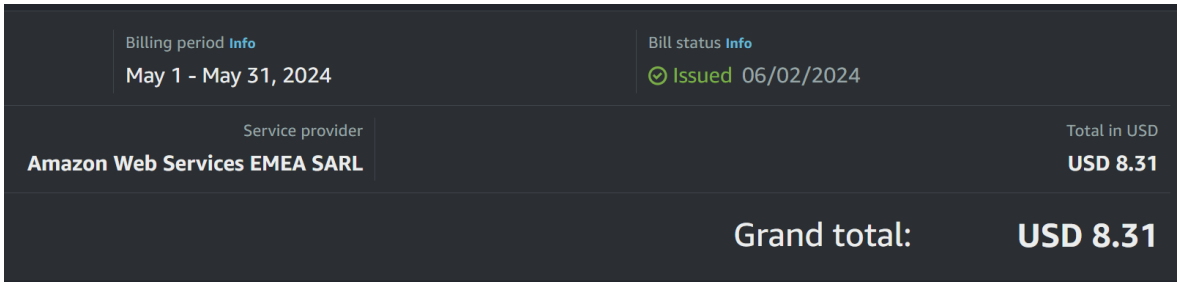


Figura 22 Factura AWS maig

Al maig, vaig esgotar el cupó de 100 euros que em va ajudar a cobrir les despeses del projecte. Com a conseqüència, la factura del mes va ascendir a 7,19 dòlars. Aquest import reflecteix el cost de l'ús dels serveis d'AWS un cop esgotat el crèdit disponible.

Pel que fa al cost d'executar la inferència utilitzant els serveis d'AWS, a continuació es detallen els preus associats als diferents serveis utilitzats:

Amazon S3: té un cost d'emmagatzematge de 0,023 dòlars dels Estats Units d'Amèrica (USD) per GB. Les peticions PUT, COPY, POST i LIST tenen un cost de 0,005 USD per cada 1.000 peticions, mentre que les peticions GET, SELECT i altres peticions costen 0,0004 USD per cada 1.000 peticions.

Amazon HealthImaging: com a part del nivell gratuït d'AWS, els nous clients d'AWS reben 20 GB d'emmagatzematge al mes, i 20.000 peticions d'API cada mes. Després d'esgotar el nivell gratuït, els costos són els següents:

- 0,105 USD per GB / mes per al nivell d'accés freqüent.
- 0,005 USD per cada 1.000 peticions d'API.

ECR: pel que fa a l'emmagatzematge a Amazon Elastic Container Registry (ECR), el cost és de 0,10 USD per GB al mes tant per a dades emmagatzemades en repositoris públics com privats.

Lambda: el nivell gratuït d'AWS Lambda inclou un milió de sol·licituds gratuïtes al mes i 400 000 GB/segons de temps de computació al mes. Per sobre d'aquest nivell, els costos són de 0,0000166667 USD per cada GB/segon de temps de computació i 0,20 USD per un milió de sol·licituds.

EC2: el cost de contractar una instància g4dn.xlarge a AWS, que inclou una GPU NVIDIA T4, 4 vCPU, 16 GiB de memòria i 1 x 125 GB d'emmagatzematge SSD NVMe, és de 0,526 USD per hora en mode de pagament sota demanda. Si es reserva la instància per un any, el cost efectiu es redueix a 0,316 USD per hora. En cas de reservar-la per tres anys, el cost efectiu és de 0,210 USD per hora.

El cost d'executar la inferència és principalment el temps que la instància EC2 està activa. Tot i que l'emmagatzematge de les imatges en S3 i HealthImaging té un cost, aquest és molt inferior al de la instància, ja que les imatges ocupen només uns quants megabytes. El cost d'emmagatzematge a ECR no es considera aquí perquè és un cost fix independentment de quantes inferències executem. Com que la imatge ocupa unes 10 GB, el cost mensual és d'aproximadament 1 USD.

En concret, la inferència triga uns 90 segons a completar-se, i tenint en compte que la instància g4dn.xlarge té un cost de 0,526 USD per hora, el cost d'executar una inferència és de 0,01315 USD, que arrodonim a 0,015 USD.

6 Legislació i protecció de dades

El codi font i el model que hem utilitzat com a exemple són els codis oficials que es poden trobar en el GitHub de MONAI. Aquest codi està disponible sota la llicència Apache 2.0, la qual cosa ens permet utilitzar, modificar i distribuir el codi d'acord amb els termes de la llicència. Aquesta llicència permet als usuaris:

- **Ús**: Utilitzar el codi font en qualsevol projecte, incloent-hi aplicacions comercials.
- **Reproducció**: Fer còpies del codi per distribuir-les.
- **Modificació**: Fer canvis en el codi font per adaptar-lo a necessitats específiques.
- **Distribució**: Redistribuir el codi original o modificat a altres usuaris.
- **Sub-llicència**: Incloure el codi en projectes que es distribueixen sota altres llicències.

Quan utilitzem codi font sota la llicència Apache 2.0, hem de complir els següents requisits per assegurar que es respecten els termes de la llicència:

- **Proporcionar una Còpia de la Llicència**: Cada vegada que redistribuïm el codi, hem d'incloure una còpia de la llicència Apache 2.0.

- **Notificar les Modificacions**: Si fem modificacions al codi original, hem d'indicar clarament aquests canvis, ja sigui a través de comentaris en el codi o mitjançant un registre de canvis.

- **Conservar Avisos de Drets d'Autor**: Hem de conservar tots els avisos de drets d'autor, patents, marques registrades i d'atribució que es troben en el codi original.

- **Incloure el Fitxer NOTICE**: Si el codi original inclou un fitxer "NOTICE", hem d'incloure una còpia llegible dels avisos d'atribució continguts en aquest fitxer en les nostres obres derivades. Això es pot fer de les següents maneres:

- **No Alterar la Llicència**: No podem modificar la llicència original del codi, però podem afegir els nostres propis termes de llicència per a les nostres modificacions o obres derivades, sempre que aquests termes no contradiguin els termes de la llicència Apache 2.0.

En el meu cas, he afegit el codi font en el meu repositori de GitHub, però no l'he modificat. Això significa que estic en les categories d'ús, reproducció i distribució del codi. He inclòs una còpia original de la llicència Apache 2.0 en el meu repositori per complir amb els requisits de la llicència. Aquests passos asseguren que es respecten els termes de la llicència i que es proporciona la informació necessària sobre els drets d'autor del codi original.

Pel que fa a la legislació espanyola, és necessari complir amb la Llei Orgànica de Protecció de Dades Personals i Garantia dels Drets Digitals (LOPD-GDD), el Reglament General de Protecció de Dades (RGPD) i la Llei de Serveis de la Societat de la Informació i Comerç Electrònic (LSSI).

En aquest context, la LOPD-GDD i el RGPD són especialment importants, ja que regulen la protecció de les dades personals i estableixen les obligacions dels responsables del tractament de dades per garantir els drets dels usuaris. Això inclou aspectes com la seguretat de les dades, el consentiment dels usuaris, la transparència en el tractament de les dades i la notificació de violacions de seguretat.

Per altra banda, la LSSI no aplica en el nostre cas perquè únicament estem executant el model al núvol i no estem fent operacions econòmiques per internet. La LSSI obliga a complir totes aquelles persones físiques o jurídiques que efectuïn operacions econòmiques per internet, com ara pàgines web que obtinguin ingressos de forma directa (venda de productes o serveis) o indirecta (publicitat en línia, bàners, patrocinis, etc.). Per tant, com que no encaixem en aquestes categories, no ens és aplicable aquesta legislació.

La protecció de dades mèdiques és un aspecte crític en el desenvolupament i desplegament d'aplicacions que gestionen informació de salut. Les dades mèdiques són especialment sensibles perquè contenen informació personal i confidencial sobre la salut i el benestar dels individus. Això fa que la protecció d'aquestes dades sigui fonamental per garantir la privacitat i seguretat dels pacients.

En el nostre cas, les dades mèdiques utilitzades en el projecte són anonimitzades i públiques. Això significa que les dades han estat transformades de manera que no es pot identificar directament als pacients, però permetent-ne el tractament per a finalitats legítimes com la recerca i el desenvolupament de models d'intel·ligència artificial.

Anonimitzar les dades implica eliminar o modificar les dades personals que podrien identificar directament a una persona, com ara noms, adreces, números d'identificació o qualsevol altra informació que pugui ser utilitzada per identificar a l'individu. Pseudonimitzar les dades, per altra banda, implica reemplaçar informació identificativa amb pseudònims o codis, que poden ser revertits sota certes condicions estrictes, però no són directament identificables.

Pel que fa als fitxers DICOM (Digital Imaging and Communications in Medicine), que són utilitzats àmpliament en imatges mèdiques, aquests poden contenir dades personals en els seus metadades. No obstant això, en aquest projecte, els fitxers DICOM utilitzats han estat anonimitzats, eliminant qualsevol informació que pugui ser utilitzada per identificar als pacients. Això assegura que les dades puguin ser compartides i processades de manera segura, complint amb les normatives de protecció de dades i garantint la privacitat dels individus.

A continuació es mostra un exemple de com es presenten les dades en les imatges que hem usat:

"EthnicGroup": "8"

"PatientBirthDate": null

"PatientID": "C3N-00198"

"PatientIdentityRemoved": "YES"

"PatientName": "C3N-00198"

"PatientSex": "M"

En aquest exemple, el nom del pacient i l'identificador del pacient han estat reemplaçats per un codi ("C3N-00198"), i s'ha indicat clarament que la identitat del pacient ha estat eliminada ("PatientIdentityRemoved": "YES"). La data de naixement del pacient ha estat nul·la ("PatientBirthDate": null) i altres dades com el grup ètnic i el sexe del pacient es mantenen, però sense la possibilitat de relacionar-los amb la identitat real del pacient.

En resum, tot i que treballem amb dades mèdiques, la informació és totalment anonimitzada, i per tant no es pot identificar a cap persona a partir dels fitxers DICOM utilitzats en el nostre projecte. Aquesta pràctica és fonamental per respectar la privacitat dels pacients i complir amb les regulacions de protecció de dades.

7 Conclusions

En conclusió, aquest projecte ha demostrat que la prova de concepte és viable i que es pot automatitzar el procés de desplegament de models d'intel·ligència artificial per a la inferència en imatges mèdiques utilitzant serveis al núvol d'AWS. Mitjançant l'ús de GitHub Actions per gestionar i automatitzar els processos de construcció, prova i desplegament, s'han obtingut bons resultats, reduint significativament el temps i els esforços necessaris per portar els models des de la fase de desenvolupament fins a la producció. Cal destacar que aquest projecte s'ha centrat principalment en el backend, i s'ha utilitzat Amazon S3 per a la gestió dels fitxers. En un entorn de producció real, seria necessari desenvolupar una interfície que permeti als usuaris pujar imatges i accedir als resultats d'una manera més intuïtiva i segura.

La part més complicada del projecte ha estat entendre com lligar tots els serveis que ofereix Amazon per aconseguir una integració eficient i funcional. La configuració dels permisos dels rols i usuaris ha estat particularment problemàtica. M'he trobat sovint amb errors indicant que no tenia accés, com ara: "User: arn:aws:iam::***/ is not authorized to perform: ...". Aquests errors s'han presentat repetidament durant el desenvolupament, requerint un temps considerable per ajustar i verificar els permisos correctes.

Aquesta dificultat es deriva de la necessitat de garantir la seguretat oferint els mínims privilegis necessaris per a cada rol/usuari, però al mateix temps assegurant que els serveis puguin interactuar correctament. Aquesta gestió de permisos és crucial per prevenir accessos no autoritzats i mantenir la integritat del sistema, però la seva configuració precisa pot ser un repte significant. Això ha posat de manifest la complexitat de gestionar de manera segura els serveis al núvol i la importància de comprendre detalladament la configuració de permisos d'AWS per evitar problemes d'autenticació i autorització.

Tot i que volia fer el procés el més autònom possible, hi ha alguns aspectes que encara requereixen intervenció manual per part de l'usuari. En concret, l'usuari ha de crear manualment els rols IAM necessaris per al funcionament de l'aplicació (només el primer cop per configurar els secrets de GitHub). Permetre que un usuari IAM (GitHub) tingui el poder de crear i gestionar rols pot portar a una mala configuració de permisos, la qual cosa podria comprometre la seguretat de tot el sistema. A més, aquest nivell d'accés podria ser explotat per executar accions no autoritzades si les credencials de l'usuari es veuen compromeses. Per tant, he optat per mantenir aquesta tasca com una responsabilitat manual, assegurant així que el control dels permisos i rols es mantingui sota supervisió directa i conscient de l'administrador del sistema.

Realitzant aquest projecte, he après moltes coses noves, incloent l'ús de GitHub Actions i la gestió de serveis al núvol amb AWS. Aquest projecte m'ha ajudat a comprendre millor com funciona el cloud computing, les seves capacitats i avantatges, així com la importància de tenir cura per no descuidar-se i evitar sorpreses en la factura. Aquesta experiència ha estat molt valuosa i enriquidora, proporcionant-me una base sòlida per a futurs projectes en el camp de la intel·ligència artificial i el núvol.

8 Referències

[1] Els models d'IA rarament arriben a producció: https://www.kdnuggets.com/2022/01/models-rarely-deployed-industrywide-failure-machine-learning-leadership.html (visitat 25-05-2024)

[2] Usar MONAI en AWS (Vídeo): https://www.nvidia.com/en-us/on-demand/session/gtcspring23-se52191/ (visitat 23-03-2024)

[3] Per què els projectes d'IA no arriben a producció: https://medium.com/@jvanlooy/7-reasons-why-80-of-ai-projects-never-make-it-to-production-5e25c2ad4d46 (visitat 25-05-2024)

[4] Per què els projectes d'IA no arriben a producció: https://d2iq.com/blog/why-87-of-ai-ml-projects-never-make-it-into-production-and-how-to-fix-it (visitat 25-05-2024)

[5] Copiar fitxer de S3 a EC2: https://stackoverflow.com/questions/64159874/copying-s3-file-to-ec2-every-time-file-posted-to-bucket (visitat 05-05-2024)

[6] Desplegar MONAI en AWS (vídeo): https://www.youtube.com/watch?v=ALuHhMYuWFk&t=1527s (visitat 15-03-2024)

[7] Repositori Github MONAI en AWS: https://github.com/aws-samples/monai-on-aws-workshop

[8] Repositori Github MONAI deploy: https://github.com/Project-MONAI/monai-deploy-app-sdk

[9] Documentació Terraform: https://registry.terraform.io/providers/hashicorp/aws/latest/docs

[10] Documentació Github Actions: https://docs.github.com/en/actions

[11] Documentació AWS: https://docs.aws.amazon.com/es_es/

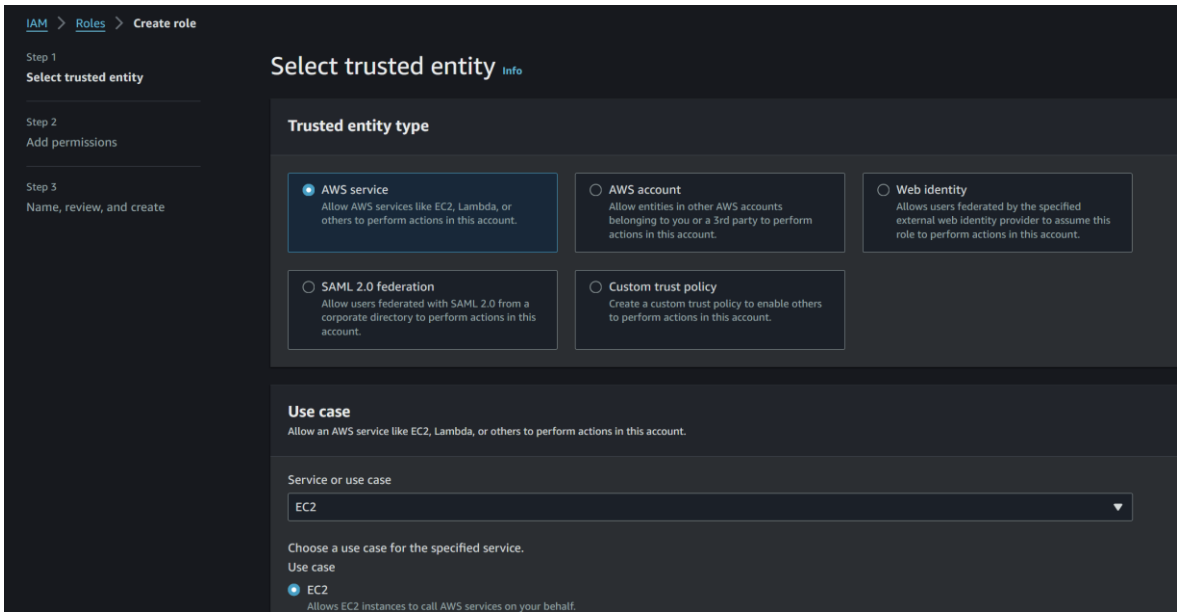
[12] Documentació MONAI: https://docs.monai.io/projects/monai-deploy-app-sdk/en/latest/

Apèndix

A Configuració AWS

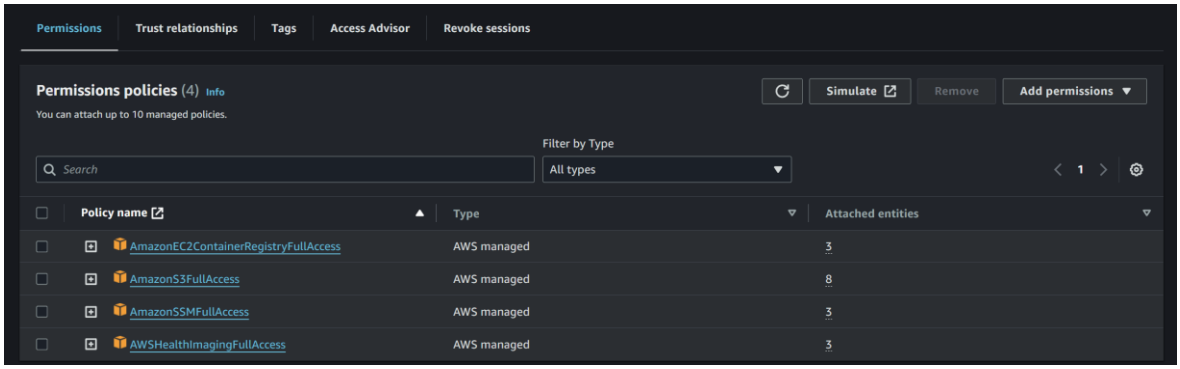
Rol EC2

Accedeix a la consola de IAM i selecciona "Roles" al menú lateral i fes clic a "Create role". Selecciona "AWS service" com a tipus d'entitat de confiança i tria "EC2" com a cas d'ús específic, ja que vols permetre que les instàncies EC2 assumeixin aquest rol.



En aquest pas, afegeix les polítiques de permisos necessàries al rol per garantir que el rol tingui les autoritzacions adequades per interactuar amb altres serveis d'AWS. Cerca i selecciona les següents polítiques gestionades per AWS:

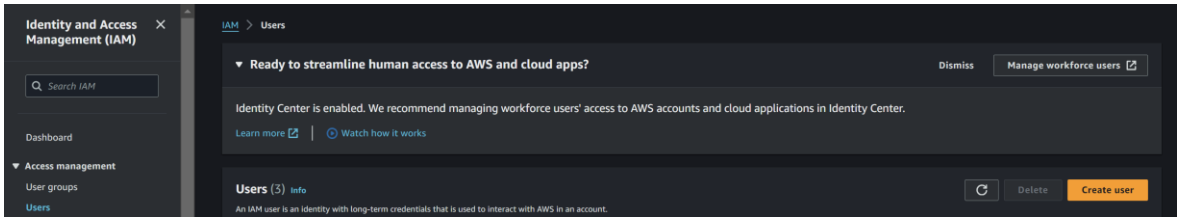
- `AmazonEC2ContainerRegistryFullAccess`: Permet accedir completament al registre de contenidors d'Amazon.
- `AmazonS3FullAccess`: Permet accedir completament als recursos d'Amazon S3.
- `AmazonSSMFullAccess`: Permet accedir completament a Amazon SSM, necessari per enviar comandes a la instància EC2.
- `AWSHealthImagingFullAccess`: Permet accedir completament als recursos d'Amazon HealthImaging.



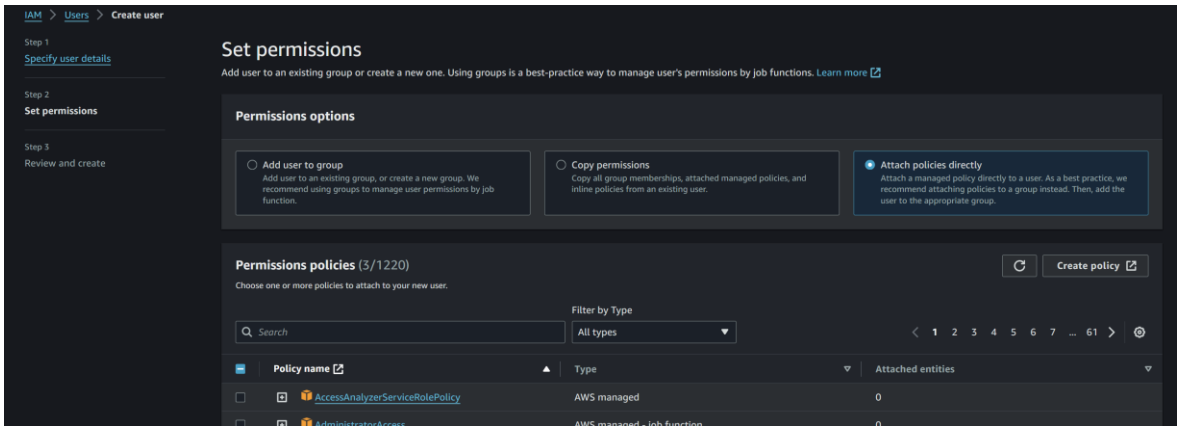
Revisa les configuracions per assegurar-te que tot estigui configurat correctament. Finalment, crea el rol assignant-li un nom que reflecteixi el seu ús i funció dins del projecte.

Usuari IAM

Ves a la consola d'AWS i selecciona el servei IAM (Identity and Access Management). Selecciona "Users" i fes clic a "Create user". Assigna un nom a l'usuari, per exemple, `github-action`.



Selecciona "Attach policies directly" per afegir polítiques de permisos. Afegiu polítiques gestionades per AWS com `AmazonEC2FullAccess`, `AmazonS3FullAccess`, `AWSLambda_FullAccess`.

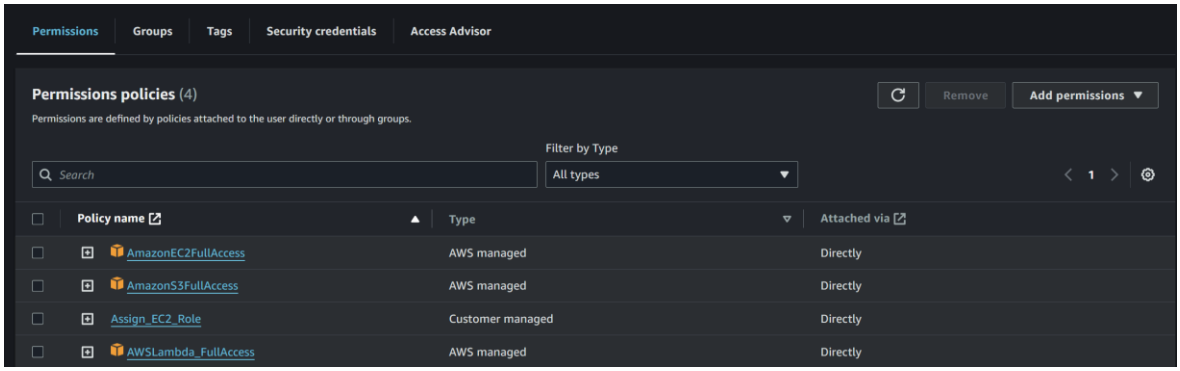


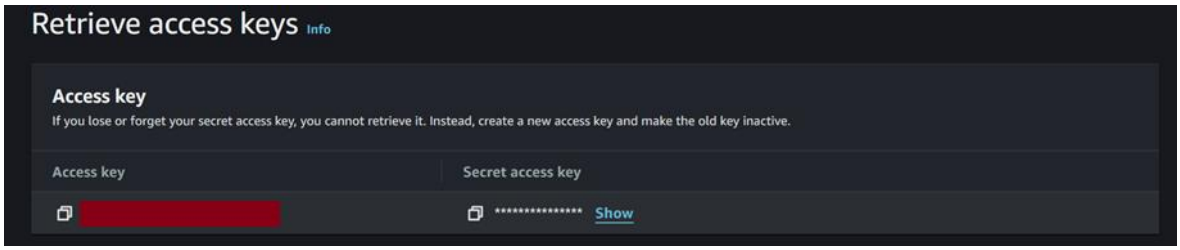
Clica el botó de “Create policy” i modifiqueu el JSON que trobaràs per defecte pel següent:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "ec2:DescribeInstances",
                "ec2:AssociateIamInstanceProfile"
            ],
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": "iam:PassRole",
            "Resource": "arn:aws:iam::$ACCOUNT_ID:role/$EC2_ROLE"
        }
    ]
}
```

Aquesta política permet a l'usuari descriure instàncies d'EC2 i associar un perfil IAM a aquestes instàncies. També permet passar un rol específic a la instància EC2. Això és més segur que donar accés complet a IAM, ja que només permet les accions necessàries.

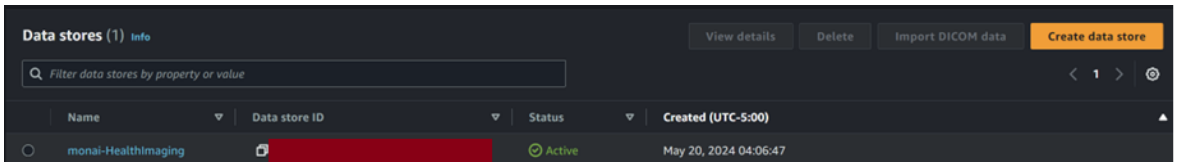
Després de crear l'usuari, selecciona l'usuari recent creat a la llista d'usuaris. Ves a la pestanya "Security credentials" i fes clic a "Create access key". Guarda el "Access key ID" i el "Secret access key" que es mostren en els secrets de github.





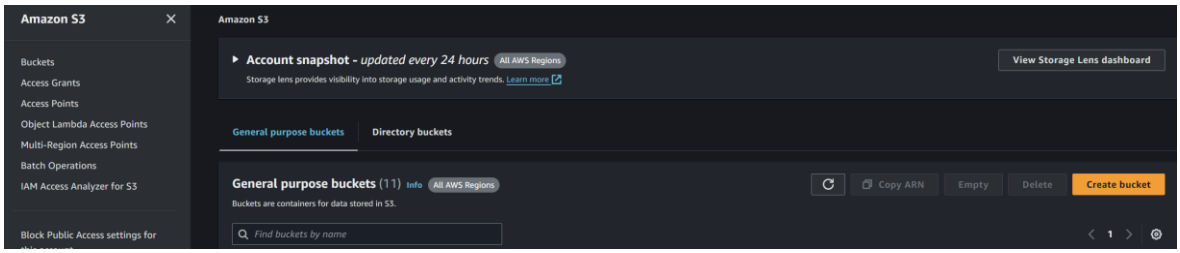
HealthImaging

A la consola de gestió d'AWS, dirigeix-te a la secció "AWS HealthImaging". Fes clic a "Create data store". Introdueix un nom per al teu data store (per exemple, `monai-HealthImaging`). Configura les opcions necessàries i fes clic a "Create".



Bucket S3

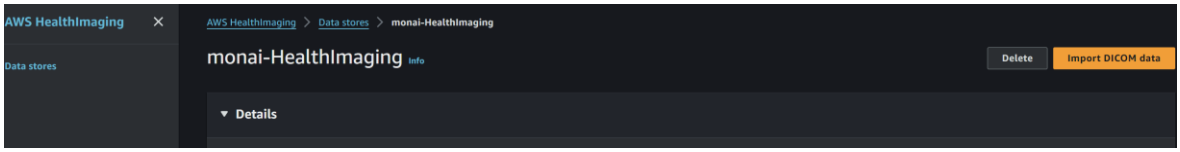
Inicia sessió a la consola d'AWS i navega a la secció de S3. A la pàgina principal de S3, fes clic a "Create bucket" per iniciar el procés de creació d'un nou bucket. Assigna un nom únic al bucket i selecciona la regió on vols emmagatzemar les dades. Configura altres opcions com la configuració de permisos, la configuració del versionat i les etiquetes si és necessari. E



Rol HealthImaging

Per crear el rol IAM necessari per a l'ús amb AWS HealthImaging, pots seguir la guia detallada proporcionada per AWS. La guia cobreix els passos necessaris per crear i configurar un rol IAM que permeti l'accés segur i adequat als serveis d'HealthImaging. Pots accedir a la guia aquí.

Opcionalment: Un cop creat el Data Store, pots importar dades DICOM directament a HealthImaging. Fes clic a "Import DICOM data" per començar. Segueix les instruccions per seleccionar les dades DICOM que vols importar. Aquest procés crearà automàticament un nou rol que podràs utilitzar.



Rol lambda

Accedeix a la consola de IAM i selecciona "Roles" al menú lateral i fes clic a "Create role". Selecciona "AWS service" com a tipus d'entitat de confiança i tria "Lambda".

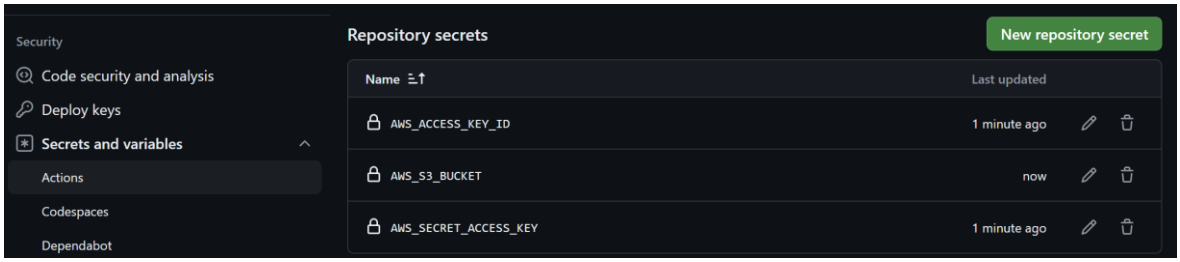
Assigna permisos per a S3 i SSM i clica el botó de “Create policy” i modifiqueu el JSON que trobaràs per defecte pel següent:

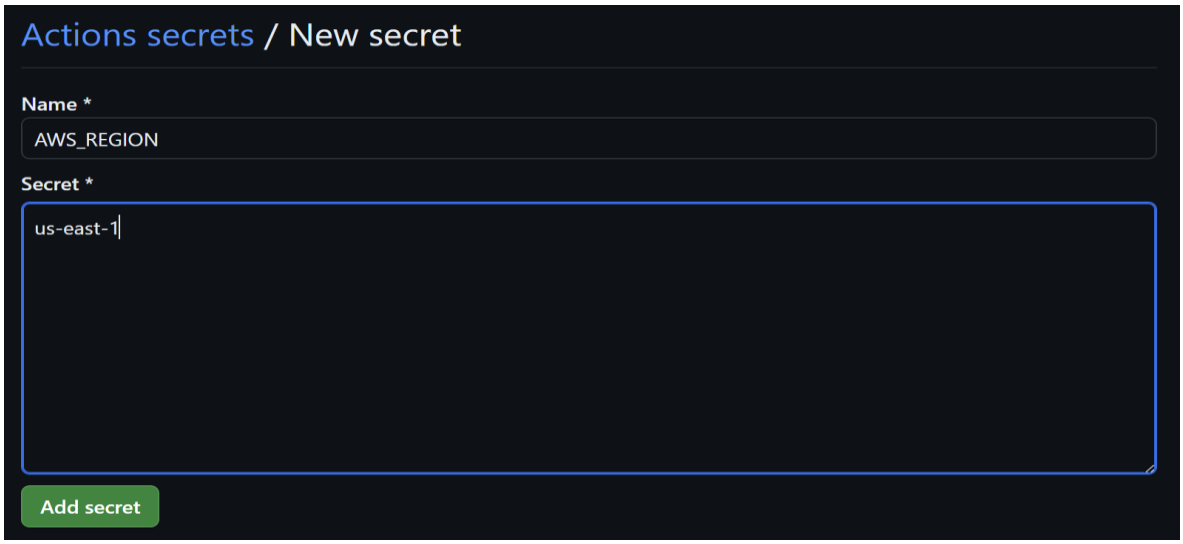
```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "logs:CreateLogGroup",
                "logs:CreateLogStream",
                "logs:PutLogEvents"
            ],
            "Resource": "arn:aws:logs:*:*:*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "ec2:Start*",
                "ec2:Stop*"
            ],
            "Resource": "*"
        }
    ]
}
```

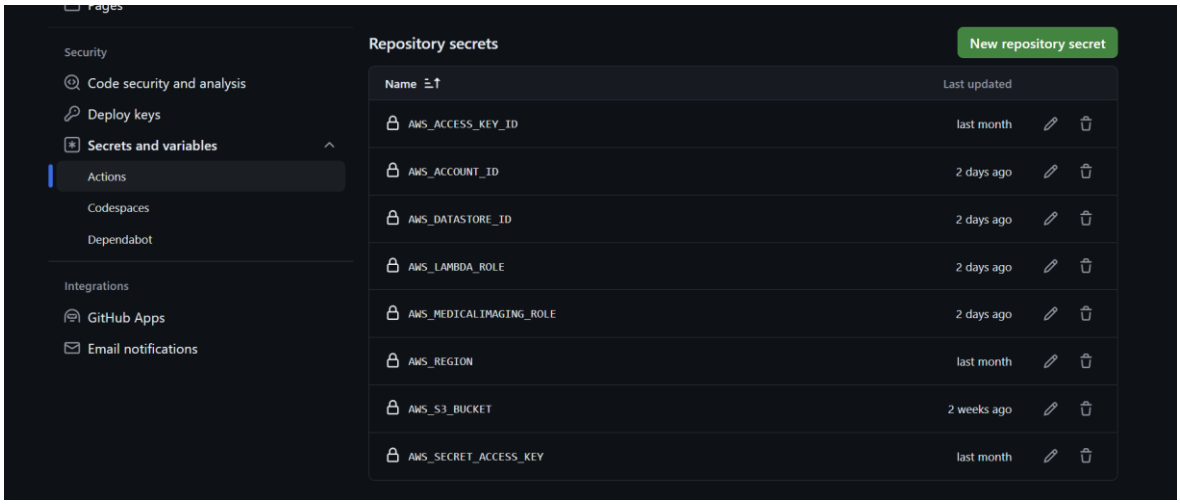
Aquesta política IAM permet al rol associat crear grups de logs, fluxos de logs i enviar esdeveniments de logs a AWS CloudWatch Logs, així com iniciar i aturar instàncies EC2. Els permisos estan dissenyats per ser prou amplis per permetre l'ús d'aquests serveis en tots els recursos i regions d'AWS.

B Configuració Github

Per configurar els secrets del repositori a GitHub, primer heu d'anar al vostre repositori i fer clic a la pestanya Settings a la part superior dreta. A continuació, a la barra lateral esquerra, seleccioneu Secrets and variables després feu clic a Actions. En acabat, feu clic al botó verd New repository secret a la part superior dreta de la pàgina per afegir un nou secret. Introduïu el nom i el valor del secret necessari. Repetiu aquest procés per a cada secret necessari. Aquest procés assegura que les credencials i altres informacions delicades es gestionen de manera segura, sent encriptades abans de ser emmagatzemades i desencriptades només al moment de l'ús en els workflows de GitHub Actions.







C Codi

action.yml

```
name: 'Create EC2 Action'
description: 'Runs a composite step action to manage EC2 and Terraform'

inputs:
  instance_name:
    required: true
    description: "Name of the EC2 instance"
  key_name:
    required: true
    description: "Name of the SSH key pair"
  ec2_role:
    required: true
    description: "Name of the IAM role to assign to the EC2 instance"
  aws_region:
    required: true
    description: "AWS Region"

outputs:
  vm_name:
    description: "VM Name"
    value: ${{ steps.get_vm_name.outputs.VM_NAME }}
  instance_id:
    description: "The ID of the EC2 instance"
    value: ${{ steps.get_instance_id.outputs.INSTANCE_ID }}
  ssh_private_key:
    description: "Path to the SSH private key file"
    value: ${{ steps.generate_ssh_key.outputs.SSH_PRIVATE_KEY_PATH }}

runs:
  using: "composite"
  steps:

    - name: Check for existing Key Pair and delete if exists
      shell: bash
      run: |
        if aws ec2 describe-key-pairs --key-names "${{ inputs.key_name
}}" ; then
            aws ec2 delete-key-pair --key-name "${{ inputs.key_name }}"
        fi

    - name: Generate SSH Key
      id: generate_ssh_key
      shell: bash
      run: |
        ssh-keygen -t rsa -f /tmp/ssh_id_gh -N ""
        echo "SSH_KEY_PATH=/tmp/ssh_id_gh.pub" >> $GITHUB_ENV
        echo "SSH_PRIVATE_KEY_PATH=/tmp/ssh_id_gh" >> $GITHUB_ENV

    - name: Setup Terraform
      uses: hashicorp/setup-terraform@v2
      with:
        terraform_wrapper: false

    - name: Initialize a new Terraform working directory
      shell: bash
```

```
      run: terraform init

    - name: Format Terraform configuration files
      shell: bash
      run: terraform fmt
    - name: Check Terraform configuration files format
      shell: bash
      run: terraform fmt -check

    - name: Terraform Apply
      shell: bash
      run: terraform apply -auto-approve -var "instance_name=${{
inputs.instance_name }}" -var "key_name=${{ inputs.key_name }}" -var
"ssh_key_path=${{ env.SSH_KEY_PATH }}" -var "aws_region=${{
inputs.aws_region }}"

    - name: Get IP address
      id: get_ip
      shell: bash
      run: |
        IP=$(terraform output -raw instance_public_ip)
        echo "AWS_IPADDRESS=$IP" >> $GITHUB_ENV
        echo "VM IP Address: $IP"
    - name: Get VM name
      id: get_vm_name
      shell: bash
      run: |
        VM_NAME="ubuntu@${{ env.AWS_IPADDRESS }}"
        echo "VM_NAME=$VM_NAME" >> $GITHUB_ENV
        echo "Constructed VM Name: $VM_NAME"

    - name: Get Instance ID
      id: get_instance_id
      shell: bash
      run: |
        INSTANCE_ID=$(terraform output -raw instance_id)
        echo "INSTANCE_ID=$INSTANCE_ID" >> $GITHUB_ENV
        echo "Instance ID: $INSTANCE_ID"

    - name: Assign IAM Role to EC2 Instance
      shell: bash
      run: |
        aws ec2 associate-iam-instance-profile --instance-id $INSTANCE_ID
--iam-instance-profile Name=${{ inputs.ec2_role }}

    - name: Test connection
      shell: bash
      run: |
        echo "Attempting to connect to: ${{ env.VM_NAME }}"
        for i in {1..10}; do
          ssh -i ${{ env.SSH_PRIVATE_KEY_PATH }} -o
StrictHostKeyChecking=no -o ConnectTimeout=10 ${{ env.VM_NAME }} "uname -
a" && break || sleep 10
        done

    - name: Set up environment / Download MONAI
      shell: bash
      run: |
        ssh -i ${{ env.SSH_PRIVATE_KEY_PATH }} ${{ env.VM_NAME }} << EOF
        virtualenv monai
        source monai/bin/activate
```

```
        pip install --upgrade monai-deploy-app-sdk
        EOF
```

create_map.yml

```
name: build_and_push_models-2
on:
  workflow_dispatch:

env:
  MONAI_APP_PATH: "${{ github.event.repository.name
}}/ai_spleen_seg_app/"
  MONAI_APP_CONFIG: "${{ github.event.repository.name
 }}/ai_spleen_seg_app/app.yaml"
  MONAI_MODEL_PATH: "${{ github.event.repository.name
 }}/ai_spleen_seg_app/model/model.ts"
  EC2_ROLE_NAME: "ec2-role"
  REPOSITORY: "monai"
  TAG: "latest"


jobs:
  setup_and_deploy:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v4

      - name: Setup AWS CLI
        run: |
          aws configure set aws_access_key_id ${{
secrets.AWS_ACCESS_KEY_ID }}
          aws configure set aws_secret_access_key ${{
secrets.AWS_SECRET_ACCESS_KEY }}
          aws configure set default.region ${{ secrets.AWS_REGION }}

      - name: Create-EC2 Action Call
        id: create-ec2
        uses: ./.github/actions/create-ec2
        with:
          instance_name: "monai-build"
          key_name: "monai-build-key"
          ec2_role: "$EC2_ROLE_NAME"
          aws_region: "${{ secrets.AWS_REGION }}"

      - name: Create-EC2 Action Output ID, VM name
        run: |
          echo "Instance ID: $INSTANCE_ID"
          echo "VM Name: $VM_NAME"

      - name: Cloning repository
        env:
          REPO_URL: ${{ github.repository }}
        run: |
          ssh -i $SSH_PRIVATE_KEY_PATH $VM_NAME "git clone --branch main
--depth 1 https://github.com/${REPO_URL}.git"

      - name: Package MONAI deploy app
        run: |
          ssh -i $SSH_PRIVATE_KEY_PATH $VM_NAME << EOF
          source monai/bin/activate
```

```
          monai-deploy package ${{ env.MONAI_APP_PATH }} \
            --config ${{ env.MONAI_APP_CONFIG }} \
            --tag seg_app:latest \
            --models ${{ env.MONAI_MODEL_PATH }} \
            --platform x64-workstation \
            -l DEBUG
          EOF

      - name: Login to AWS ECR
        run: |
           ssh -i $SSH_PRIVATE_KEY_PATH $VM_NAME "aws ecr get-login-
password --region ${{ secrets.AWS_REGION }} | docker login --username AWS
--password-stdin ${{ secrets.AWS_ACCOUNT_ID }}.dkr.ecr.${{
secrets.AWS_REGION }}.amazonaws.com"

      - name: Check and create ECR repository if not exists
        run: |
           ssh -i $SSH_PRIVATE_KEY_PATH $VM_NAME "
             if ! aws ecr describe-repositories --region ${{
secrets.AWS_REGION }} --repository-names ${{ env.REPOSITORY }} >
/dev/null 2>&1; then
               echo 'Repository ${{ env.REPOSITORY }} does not exist.
Creating it...'
               aws ecr create-repository --repository-name ${{
env.REPOSITORY }} --region ${{ secrets.AWS_REGION }}
             else
               echo 'Repository ${{ env.REPOSITORY }} already exists.'
             fi
           "

      - name: Tag Docker Image
        run: |
          DOCKER_IMAGE="${{ secrets.AWS_ACCOUNT_ID }}.dkr.ecr.${{
secrets.AWS_REGION }}.amazonaws.com/${REPOSITORY}:${TAG}"
          ssh -i $SSH_PRIVATE_KEY_PATH $VM_NAME "docker tag seg_app-x64-
workstation-dgpu-linux-amd64:latest $DOCKER_IMAGE"
          echo "DOCKER_IMAGE=$DOCKER_IMAGE" >> $GITHUB_ENV

      - name: Push Docker Image
        run: |
          ssh -i $SSH_PRIVATE_KEY_PATH $VM_NAME "docker push
$DOCKER_IMAGE"

      - name: Terminate EC2 Instance
        run: |
          aws ec2 terminate-instances --instance-ids $INSTANCE_ID --
region ${{ secrets.AWS_REGION }}
          echo "Waiting for termination to complete..."
          aws ec2 wait instance-terminated --instance-ids $INSTANCE_ID --
region ${{ secrets.AWS_REGION }}
          echo "Instance $INSTANCE_ID terminated."
```

configure_inference.yml

```
name: configure_inference
on:
  workflow_dispatch:

env:
  FUNCTION_NAME: "LambdaMonai"
  EC2_ROLE_NAME: "ec2-role"
  REPOSITORY: "monai"
  TAG: "latest"

jobs:
  setup_inference:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v4

      - name: Setup AWS CLI
        run: |
          aws configure set aws_access_key_id ${{
secrets.AWS_ACCESS_KEY_ID }}
          aws configure set aws_secret_access_key ${{
secrets.AWS_SECRET_ACCESS_KEY }}
          aws configure set default.region ${{ secrets.AWS_REGION }}

      - name: Create-EC2 Action Call
        id: create-ec2
        uses: ./.github/actions/create-ec2
        with:
          instance_name: "monai-run"
          key_name: "monai-run-key"
          ec2_role: "$EC2_ROLE_NAME"
          aws_region: "${{ secrets.AWS_REGION }}"

      - name: Create-EC2 Action Output ID, VM name
        run: |
          echo "Instance ID: $INSTANCE_ID"
          echo "VM Name: $VM_NAME"

      - name: Login to AWS ECR
        run: |
           ssh -i $SSH_PRIVATE_KEY_PATH $VM_NAME "aws ecr get-login-
password --region ${{ secrets.AWS_REGION }} | docker login --username AWS
--password-stdin ${{ secrets.AWS_ACCOUNT_ID }}.dkr.ecr.${{
secrets.AWS_REGION }}.amazonaws.com"

      - name: Docker pull
        run: |
           ssh -i $SSH_PRIVATE_KEY_PATH $VM_NAME "docker pull ${{
secrets.AWS_ACCOUNT_ID }}.dkr.ecr.${{ secrets.AWS_REGION
}}.amazonaws.com/${{ env.REPOSITORY }}:${{ env.TAG }}"

      - name: Update setup.sh with secrets
        run: |
          cd setup
          sed -i 's|ROLE_ARN=""|ROLE_ARN="${{ secrets.AWS_LAMBDA_ROLE
}}"|' setup.sh
```

```
          sed -i 's|REGION=""|REGION="${{ secrets.AWS_REGION }}"|'
setup.sh
          sed -i 's|BUCKET_NAME=""|BUCKET_NAME="${{ secrets.AWS_S3_BUCKET
}}"|' setup.sh
          sed -i 's|ACCOUNT_ID=""|ACCOUNT_ID="${{ secrets.AWS_ACCOUNT_ID
}}"|' setup.sh
          sed -i 's|FUNCTION_NAME=""|FUNCTION_NAME="${{ env.FUNCTION_NAME
}}"|' setup.sh

      - name: Update ec2.sh with secrets
        run: |
          cd setup
          sed -i 's|ACCOUNT_ID=""|ACCOUNT_ID="${{ secrets.AWS_ACCOUNT_ID
}}"|' ec2.sh
          sed -i 's|ROLE_ARN=""|ROLE_ARN="${{
secrets.AWS_MEDICALIMAGING_ROLE }}"|' ec2.sh
          sed -i 's|DATASTORE_ID=""|DATASTORE_ID="${{
secrets.AWS_DATASTORE_ID }}"|' ec2.sh
          sed -i 's|REGION=""|REGION="${{ secrets.AWS_REGION }}"|' ec2.sh
          sed -i 's|DOCKER_IMAGE_TAG=""|DOCKER_IMAGE_TAG="${{
env.REPOSITORY }}:${{ env.TAG }}"|' ec2.sh

      - name: Update lambda_function.py with instance ID
        run: |
          cd setup
          sed -i 's|region = ""|region = "${{ secrets.AWS_REGION }}"|'
lambda_function.py
          sed -i 's|static_instance_id = ""|static_instance_id = "${{
env.INSTANCE_ID }}"|' lambda_function.py

      - name: Send ec2.sh to EC2 instance
        run: |
          scp -i  $SSH_PRIVATE_KEY_PATH -o StrictHostKeyChecking=no
setup/ec2.sh $VM_NAME:/home/ubuntu/ec2.sh
          ssh -i $SSH_PRIVATE_KEY_PATH $VM_NAME "chmod +x
/home/ubuntu/ec2.sh"

      - name: Run setup.sh
        run: |
          cd setup
          chmod +x setup.sh
          ./setup.sh
```

ec2.sh

```
#!/bin/bash
LOG_FILE="/home/ubuntu/script.log"

# Redirect stdout and stderr to the log file
exec > >(tee -a $LOG_FILE) 2>&1

# Define the parameter names
BUCKET_NAME_PARAM="s3_bucket_name"
OBJECT_KEY_PARAM="s3_object_key"

# Define variables
ACCOUNT_ID=""
ROLE_ARN=""
DATASTORE_ID=""
REGION=""
DOCKER_IMAGE_TAG=""

# Set the AWS region
aws configure set region $REGION

BUCKET_NAME=$1
OBJECT_KEY=$2

# Define the local destination paths
INPUT_PATH="/home/ubuntu/input"
OUTPUT_PATH="/home/ubuntu/output"
mkdir -p $INPUT_PATH
mkdir -p $OUTPUT_PATH

# Clean up any existing files in the input and output directories
sudo rm -rf $INPUT_PATH/*
sudo rm -rf $OUTPUT_PATH/*

# Download the file from S3
aws s3 cp "s3://$BUCKET_NAME/$OBJECT_KEY" "$INPUT_PATH/$(basename
$OBJECT_KEY)"

# Verify the download
if [ -f "$INPUT_PATH/$(basename $OBJECT_KEY)" ]; then
  echo "File downloaded successfully to $INPUT_PATH"
else
  echo "Failed to download the file from S3"
  exit 1
fi

# Check if the downloaded file is a zip file
FILE_EXTENSION="${OBJECT_KEY##*.}"
if [ "$FILE_EXTENSION" == "zip" ]; then
  echo "The file is a zip file. Unzipping..."
  unzip "$INPUT_PATH/$(basename $OBJECT_KEY)" -d $INPUT_PATH
  rm "$INPUT_PATH/$(basename $OBJECT_KEY)"
  if [ $? -ne 0 ]; then
    echo "Failed to unzip the file"
    exit 1
  fi
fi


# Docker login and pull the latest image
```

```
aws ecr get-login-password --region $REGION | docker login --username AWS
--password-stdin $ACCOUNT_ID.dkr.ecr.$REGION.amazonaws.com
docker pull $ACCOUNT_ID.dkr.ecr.$REGION.amazonaws.com/$DOCKER_IMAGE_TAG

# Run monai-deploy
source /home/ubuntu/monai/bin/activate
monai-deploy run
$ACCOUNT_ID.dkr.ecr.$REGION.amazonaws.com/$DOCKER_IMAGE_TAG -i
$INPUT_PATH -o $OUTPUT_PATH

BASE_NAME=$(basename "$OBJECT_KEY" .zip)

# Upload the results to S3 under /results/$BASE_NAME/
aws s3 cp $OUTPUT_PATH s3://$BUCKET_NAME/results/$BASE_NAME/ --recursive

# Check if the file is a zip file before uploading and deleting
if [ "$FILE_EXTENSION" == "zip" ]; then
  # Upload the input files to S3 under /inferred/
  aws s3 cp $INPUT_PATH s3://$BUCKET_NAME/inferred/$BASE_NAME/ --
recursive

  # Delete the downloaded zip from S3 input bucket
  aws s3 rm s3://$BUCKET_NAME/$OBJECT_KEY
else
  # Move the file within S3 from /input to /inferred
  aws s3 mv s3://$BUCKET_NAME/$OBJECT_KEY
s3://$BUCKET_NAME/inferred/$(basename $OBJECT_KEY)
fi


# Start the DICOM import job for results
aws medical-imaging start-dicom-import-job \
    --job-name "my-dicom-import-job" \
    --datastore-id "$DATASTORE_ID" \
    --data-access-role-arn "$ROLE_ARN" \
    --input-s3-uri "s3://$BUCKET_NAME/results/$BASE_NAME/" \
    --output-s3-uri "s3://$BUCKET_NAME/HealthImaging/"

attempt=1
max_attempts=10
while [ $attempt -le $max_attempts ]; do
        # Start the DICOM import job for infered files
        result=$( aws medical-imaging start-dicom-import-job \
                  --job-name "my-dicom-import-job" \
                  --datastore-id "$DATASTORE_ID" \
                  --data-access-role-arn "$ROLE_ARN" \
                  --input-s3-uri "s3://$BUCKET_NAME/input/$BASE_NAME/" \
                  --output-s3-uri "s3://$BUCKET_NAME/HealthImaging/"
2>&1)

        if [[ $result == *"Too Many Requests"* ]]; then
            retry=$((attempt * 10))
            echo "Throttling exception encountered. Retrying in $retry
seconds..."
            sleep $retry
            attempt=$((attempt + 1))
        else
            echo "DICOM import job started successfully."
            break
        fi
    done
```

lambda_function.py

```
import boto3
import json
import logging

# Set up logging
logger = logging.getLogger()
logger.setLevel(logging.INFO)

region = ""
static_instance_id = ""

ec2 = boto3.client('ec2', region_name=region)
ssm = boto3.client('ssm', region_name=region)

def lambda_handler(event, context):
    try:
        # Extract bucket name and object key from the event
        bucket_name = event['Records'][0]['s3']['bucket']['name']
        object_key = event['Records'][0]['s3']['object']['key']

        # Use the static instance ID
        instance_id = static_instance_id

        # Check the current state of the instance
        response = ec2.describe_instances(InstanceIds=[instance_id])
        instance_state =
response['Reservations'][0]['Instances'][0]['State']['Name']

        if instance_state != 'running':
            # Start EC2 instance
            ec2.start_instances(InstanceIds=[instance_id])
            logger.info('Starting your instance: %s', instance_id)

            # Wait until the instance is running
            waiter = ec2.get_waiter('instance_running')
            waiter.wait(InstanceIds=[instance_id])
            logger.info('Instance is now running')
        else:
            logger.info('Instance is already running')

        # Update SSM Parameter Store
      #  ssm.put_parameter(
      #      Name='s3_bucket_name',
      #      Value=bucket_name,
      #      Type='String',
      #      Overwrite=True
      #  )

      #  ssm.put_parameter(
      #      Name='s3_object_key',
      #      Value=object_key,
      #      Type='String',
      #      Overwrite=True
      #  )

      #  logger.info('S3 bucket name and object key stored in SSM
Parameter Store')

        # Send command to EC2 instance to execute script with arguments
        response = ssm.send_command(
```

```
            InstanceIds=[instance_id],
            DocumentName="AWS-RunShellScript",
            Parameters={
                'commands': [
                    f'sudo su - ubuntu -c "/home/ubuntu/ec2.sh
{bucket_name} {object_key}"'
                ]
            }
        )
        logger.info('SSM Run Command sent to instance: %s with bucket: %s
and object: %s', instance_id, bucket_name, object_key)


        return {
            'statusCode': 200,
            'body': json.dumps('SSM parameters updated and command sent
to EC2 instance.')
        }
    except Exception as e:
        logger.error('Error: %s', str(e))
        return {
            'statusCode': 500,
            'body': json.dumps(f"Error: {str(e)}")
        }
```

setup.sh

```
#!/bin/bash

# Variables
FUNCTION_NAME=""
ROLE_ARN=""
REGION=""
BUCKET_NAME=""
ACCOUNT_ID=""

# Zip the Lambda function code
zip -r function.zip lambda_function.py

# Create or update the Lambda function
aws lambda create-function --function-name $FUNCTION_NAME \
--zip-file fileb://function.zip --handler lambda_function.lambda_handler
\
--runtime python3.8 --role $ROLE_ARN --region $REGION \
|| \
aws lambda update-function-code --function-name $FUNCTION_NAME \
--zip-file fileb://function.zip --region $REGION

# Add S3 bucket permissions to invoke the Lambda function
aws lambda add-permission --function-name $FUNCTION_NAME --statement-id
S3Invoke --action lambda:InvokeFunction --principal s3.amazonaws.com --
source-arn arn:aws:s3:::$BUCKET_NAME

# Create S3 event configuration JSON
cat <<EOT > s3_event_configuration.json
{
  "LambdaFunctionConfigurations": [
    {
      "LambdaFunctionArn":
"arn:aws:lambda:$REGION:$ACCOUNT_ID:function:$FUNCTION_NAME",
      "Events": ["s3:ObjectCreated:*"],
      "Filter": {
        "Key": {
          "FilterRules": [
            {
              "Name": "prefix",
              "Value": "input/"
            }
          ]
        }
      }
    }
  ]
}
EOT

# Apply the S3 event configuration
aws s3api put-bucket-notification-configuration --bucket $BUCKET_NAME --
notification-configuration file://s3_event_configuration.json

echo "Lambda function and S3 event notification configured successfully."
```

ami.sh

```
#!/bin/bash

# Install pip
sudo apt update
sudo apt install -y python3-pip python3.12-venv

# Install Docker
# https://docs.docker.com/engine/install/ubuntu/
sudo apt-get update
sudo apt-get install -y ca-certificates curl
sudo install -m 0755 -d /etc/apt/keyrings
sudo curl -fsSL https://download.docker.com/linux/ubuntu/gpg -o
/etc/apt/keyrings/docker.asc
sudo chmod a+r /etc/apt/keyrings/docker.asc

echo \
  "deb [arch=$(dpkg --print-architecture) signed-
by=/etc/apt/keyrings/docker.asc] 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-get update

sudo apt-get install -y docker-ce docker-ce-cli containerd.io docker-
buildx-plugin docker-compose-plugin
sudo usermod -aG docker $USER

# Install CUDA toolkit and container toolkit
# https://docs.nvidia.com/datacenter/cloud-native/container-
toolkit/latest/install-guide.html
sudo apt install -y nvidia-cuda-toolkit

curl -fsSL https://nvidia.github.io/libnvidia-container/gpgkey | sudo gpg
--dearmor -o /usr/share/keyrings/nvidia-container-toolkit-keyring.gpg
curl -s -L https://nvidia.github.io/libnvidia-
container/stable/deb/nvidia-container-toolkit.list | \
    sed 's#deb https://#deb [signed-by=/usr/share/keyrings/nvidia-
container-toolkit-keyring.gpg] https://#g' | \
    sudo tee /etc/apt/sources.list.d/nvidia-container-toolkit.list

sudo apt-get update
sudo apt-get install -y nvidia-container-toolkit

sudo apt install -y nvidia-utils-535
sudo apt install -y nvidia-driver-535
sudo apt install -y nvidia-container-runtime

sudo tee /etc/docker/daemon.json > /dev/null <<EOL
{
  "default-runtime": "nvidia",
  "runtimes": {
    "nvidia": {
      "path": "nvidia-container-runtime",
      "runtimeArgs": []
    }
  }
}
EOL
```

```
# Install libcudart11.0
# https://developer.nvidia.com/cuda-11.0-download-archive?target_os=Linux
wget
https://developer.download.nvidia.com/compute/cuda/repos/ubuntu2004/x86_6
4/cuda-ubuntu2004.pin
sudo mv cuda-ubuntu2004.pin /etc/apt/preferences.d/cuda-repository-pin-
600
wget
http://developer.download.nvidia.com/compute/cuda/11.0.2/local_installers
/cuda-repo-ubuntu2004-11-0-local_11.0.2-450.51.05-1_amd64.deb
sudo dpkg -i cuda-repo-ubuntu2004-11-0-local_11.0.2-450.51.05-1_amd64.deb
sudo apt-key add /var/cuda-repo-ubuntu2004-11-0-local/7fa2af80.pub
sudo apt-get update
sudo apt-get -y install cuda
rm cuda-repo-ubuntu2004-11-0-local_11.0.2-450.51.05-1_amd64.deb

# Install AWS CLI
# https://docs.aws.amazon.com/cli/latest/userguide/getting-started-
install.html
sudo apt install -y unzip
curl "https://awscli.amazonaws.com/awscli-exe-linux-x86_64.zip" -o
"awscliv2.zip"
unzip awscliv2.zip
sudo ./aws/install
rm awscliv2.zip

# Install Anaconda dependencies and Anaconda itself
# https://docs.anaconda.com/free/anaconda/install/linux/
sudo apt-get install -y libgl1-mesa-glx libegl1-mesa libxrandr2
libxrandr2 libxss1 libxcursor1 libxcomposite1 libasound2 libxi6 libxtst6
curl -O https://repo.anaconda.com/archive/Anaconda3-2024.02-1-Linux-
x86_64.sh
sudo bash Anaconda3-2024.02-1-Linux-x86_64.sh -b -p $HOME/anaconda3
echo 'export PATH="$HOME/anaconda3/bin:$PATH"' >> ~/.bashrc
rm Anaconda3-2024.02-1-Linux-x86_64.sh
sudo chown -R ubuntu:ubuntu /home/ubuntu/anaconda3
source ~/.bashrc

# Create a virtual environment
conda update -n base -c defaults conda -y
conda create -n monai python=3.8 pytorch torchvision  cudatoolkit=11.0 -c
pytorch -c conda-forge --yes
conda init bash
exec $SHELL
conda activate monai
pip install monai-deploy-app-sdk scikit-image matplotlib holoscan gdown

sudo reboot
```