UNIVERSITAT POLITÈCNICA DE VALÈNCIA

TELECOM ESCUELA TÉCNICA VLC SUPERIOR DE INGENIERÍA DE TELECOMUNICACIÓN

Diseño, desarrollo e implementación de un protocolo automático para la gestión inteligente de tareas en una infraestructura de inteligencia artificial

David Pons Moro

Tutor: Valery Naranjo Ornedo

Cotutor: Cristian Pulgarín Ospina

Cotutor: Adrián Colomer Granero

Trabajo Fin de Grado presentado en la Escuela Técnica Superior de Ingeniería de Telecomunicación de la Universitat Politècnica de València, para la obtención del Título de Graduado en Ingeniería de Tecnologías y Servicios de Telecomunicación

Curso 2020-21

Valencia, 18 de Septiembre de 2021

Resumen

Uno de los problemas de tener un *cluster* especializado en el entrenamiento de inteligencia artificial es el de realizar una gestión óptima de los recursos necesarios para el envío de tareas, ya que dicho proceso resulta poco eficiente si no dispone de un programa especializado para ello, es por eso que en el presente trabajo final de grado se aborda la implementación de un sistema de colas para un *cluster* producto de la unión de servidores dedicados al entrenamiento de modelos de *deep learning*, de este modo dejamos de ver los servidores por separado a un único servidor en el que encolar las tareas. El sistema que se implementa tiene entre otras ventajas la automatización del envió de los *jobs* una vez puestos en cola, ya que este sistema será capaz de elegir a que servidor enviar cada trabajo de entrenamiento teniendo en cuenta los requerimientos de dicho trabajo y la disponibilidad de recursos en ese instante temporal. Adicionalmente, el protocolo también contará con calidad de servicio en las colas con lo que se puede variar las prioridades en las mismas.

Resum

Un dels problemes de tindre un *clúster* especialitzat en l'entrenament d'intel·ligència artificial és el de realitzar una gestió òptima dels recursos necessaris per a l'enviament de tasques, ja que aquest procés resulta poc eficient si no disposa d'un programa especialitzat per a això, és per això que en el present treball final de grau s'aborda la implementació d'un sistema de cues per a un *clúster* producte de la unió de servidors dedicats a l'entrenament de models de *deep learning*, d'aquesta manera deixem de veure els servidors per separat a un únic servidor en el qual encolar les tasques. El sistema que s'implementa té entre altres avantatges l'automatització de l'enviament dels *jobs* un cop posats en cua, ja que aquest sistema serà capaç de triar a què servidor enviar cada treball d'entrenament tenint en compte els requeriments de l'esmentat treball i la disponibilitat de recursos en aquest instant temporal. Addicionalment, el protocol també comptarà amb qualitat de servei en les cues amb el que es pot variar les prioritats en les mateixes.

Abstract

One of the problems of having a specialized cluster in artificial intelligence (AI) training is to perform optimal management of the required resources to accomplish a specific task. This process is inefficient if specialized queue-manager tools are not used. This is the main motivation of this final degree project in which the implementation of a smart protocol for task queue management is designed, implemented and deployed in a real AI cluster environment. This cluster is composed of several servers dedicated to training deep learning-based models. In this way, servers go from being used individually to being part of a single wide server in which to queue the tasks. The implemented protocol has among other advantages the automation of the job sending process once they are queued. The resulting system is capable of choosing the optimal individual server to send each training job taking into account the requirements of the current job and the availability of resources in that specific timestamp. Additionally, the quality of service in the queues is analysed by the designed protocol, which can result in the variation of the queue priorities.

Índice general

Índice de figuras

Índice de tablas

Listado de siglas empleadas

CNN *Convolutional Neural Network*.

CPU *Central Processing Unit*.

GPU *Graphics Processing Unit*.

IA Inteligencia Artificial.

TFG Trabajo Final de Grado.

Parte I

Memoria

Capítulo 1

Introducción

1.1. Justificación y contexto del trabajo

Las GPU han ganado importancia en múltiples ámbitos del sector tecnológico con usos tan variados como la ejecución de juegos los cuales cuentan cada vez con mayor realismo y por lo tanto mayor nivel de detalles a procesar , tareas ligadas a la IA o minado de criptomonedas entre otras, dichas tareas hacen un buen uso de esa alta potencia de calculo que las GPU ofrecen y tratan de aprovechar al máximo la ventaja del paralelismo que ofrecen.

Entrando en detalle con la IA, se han ido produciendo diversos avances en este campo desde que se dio a conocer por primera vez, ejemplo de esto se puede encontrar en la evolución en la aplicación relacionada con el juego y es que en 1997 *Gari Kaspárov*, campeón mundial de ajedrez, perdió frente al ordenador autónomo Deep Blue [1] siendo la primera inteligencia artificial en conseguirlo, desde entonces se han producido diversos avances llegando al presentado de la mano de *OpenAI*, este mediante el uso de aprendizaje reforzado multiagente consigue que una serie de *bots* divididos en dos equipos, los que se tienen que esconder y los que tienen que pillarlos, jueguen al escondite entre ellos con una serie de restricciones y normas preestablecidas, lo cual hace que desarrollen constantemente nuevas estrategias para lograr su cometido [2], esta misma compañía también hizo uso de esta técnica de aprendizaje en otro gran logro, este vez consistía en vencer a los campeones del mundo de Dota2 mediante *bots* entrenados durante el equivalente a 180 años de partidas jugadas [3], dicho juego para ordenadores requiere de una gran coordinación entre los miembros del mismo equipo, donde cada personaje juega un papel diferente, y desarrollar estrategias complejas que permitan defender la propia base y atacar la del rival, el mapa donde se desarrolla dicha acción cuenta con múltiples caminos que unen ambas bases, adicionalmente ambos equipos se encuentran con diferentes obstáculos incluidos los propios integrantes del equipo rival, a este genero de juego se le conoce como MOBA y se puden encontrar variantes como el *Smite* o *League Of Legend* (LOL).

Dejando de lado el sector del juego también se pueden encontrar grandes avances en diversos campos de la IA, entre ellos el proyecto *Atlas* perteneciente al campo de la robótica el cual consiste en el desarrollo de un robot bípedo capaz de sortear obstáculos irregulares llegando a hacer movimientos de *parkour* [4], en el campo de la salud mediante el análisis de casos ligados al Alzheimer [5], en el campo de la visión artificial consiguiendo diferenciar y colorear figuras de un vídeo en blanco y negro [6] , o en el campo de la lingüística mejorando las estructuras de los algoritmos

para la traducción [7].

Todos estos avances, cada vez mas sotisficados, requieren de grandes cantidades de datos que serán analizados de forma simultanea y en paralelo, es por esto que se unan unidades de procesamiento gráfico como pieza fundamental ya que según se ha podido comprobar estas unidades han resultado más efectivas durante estos últimos años a la hora de ejecutar tareas de inteligencia artificial frente a las CPU [8].

Es por esto que para el entrenamiento de tareas de inteligencia artificial exigentes se cuenta con una agrupación de GPU's llamada *cluster* de GPU con la cual se consigue unificar los recursos pudiéndolos orquestar de forma correcta para la ejecución de estas tareas.

Es aquí donde se plantea la necesidad de contar con un protocolo que de forma automática gestione las tareas que son enviadas al *cluster* de GPU, estos tipos de protocolos son una parte clave dentro de los *clusters*, ya que no solo gestionan el envío de tareas a los diferentes nodos, si no que también tienen funcionalidades tan importantes como el aprovisionamiento de recursos que serán necesarios para la ejecución de tareas, en este caso de inteligencia artificial, especificando campos como el número de GPU necesarias, la memoria máxima, o el tiempo de ejecución. Además estos protocolos cuentan también con funcionalidades como el balanceo de cargas, con el cual se consigue optimizar los recursos disponibles del *cluster* y se evita también la saturación o descompensación de trabajo de ciertos nodos.

Es por esto que en este trabajo se aborda el diseñar, desarrollar e implementar un protocolo automático para la gestión inteligente de tareas en una infraestructura de inteligencia artificial con el fin de optimizar el uso de dicha infraestructura.

1.2. Marco de trabajo

Este TFG está enmarcado dentro del grupo de investigación CVBLab (*Computer Vision and Behavoir Analysis Lab*) el cual está ubicado en la Universidad Politecnica de Valencia.

Dicho grupo, dirigido por la catedrática Valery Naranjo Ornedo, es experto en el desarrollo y aplicación de nuevas metodologías en inteligencia artificial a distintos sectores industriales. Una contrastada experiencia en el tratamiento de señal, imagen, vídeo y texto dotan al grupo de investigación de una alta capacidad para generar modelos de predicción empleando cualquier tipo de datos. En una era en la que minimizar los costes, aumentar la seguridad y reducir la carga de trabajo de especialistas son una prioridad, las soluciones basadas en inteligencia artificial son cada vez más demandadas. Particularmente, el grupo CVBLab focaliza sus esfuerzos en proveer soluciones ante distintos escenarios basadas en el aprendizaje máquina a partir de grandes cantidades de datos. Mediante el diseño y entrenamiento de redes neuronales profundas (i.e. *Deep Learning*) los investigadores del grupo dan solución a problemas de clasificación, regresión, generación de contenido, detección/segmentación de objetos, descripción automática de escenas, interacción automática entorno-agente, procesamiento natural del lenguaje y un largo etc. haciendo uso de paradigmas tanto supervisados, como no supervisados, así como basados en el aprendizaje por refuerzo.

Debido a la complejidad computacional de las técnicas que se desarrollan en el grupo de investigación, CVBLab ha diseñado una infraestructura de computación de altas prestaciones. El actor principal de esta infraestructura de inteligencia artificial es el sistema NVIDIA DGX A100. Dicha

máquina está compuesta por 8x NVIDIA A100 Tensor Core GPUs ofreciendo 5 petaFLOPS de potencia de cálculo y 320GB de memoria RAM. Adicionalmente, CVBLab dispone de cuatro equipos Intel i7 @4.20GHz con 32GB de RAM y (2x) tarjetas gráficas NVIDIA Titan XP y dos equipos Intel i7 @4.20GHz con 32GB de RAM y (2x) tarjetas gráficas NVIDIA Titan V en cada máquina. El grupo también dispone de dos servidores NAS (Synology DS416 y Synology DS918+) en los que se alojan los datos y la documentación de los diferentes proyectos que se desarrollan con 32 y 16 TB de capacidad, respectivamente.

1.3. Estado del arte

Las GPU están siendo utilizadas tanto en el campo del *gaming* como en el procesamiento de datos, esto es debido a la gran cantidad de cálculos que de forma paralela y simultanea pueden realizar, es por eso y junto a su inicial bajo costo años atrás que han tenido un gran crecimiento en estos últimos años en cuanto a rendimiento y popularidad se refiere, centrándonos en el rendimiento se está consiguiendo que cada 6 meses se duplique siendo este crecimiento mayor al pronosticado según la ley de *Moore* para las CPU, ley la cual sostiene que el rendimiento de estas se duplicaría cada 2 años aproximadamente.[9][10].

Prueba de este crecimiento se puede encontrar en los nuevos modelos de *Nvidia* orientados al campo de inteligencia artificial tales como Tesla V100 diseñado para *Data Centers*, Drive Pegasus la cual está ideada para sistemas autónomos como el caso de la conducción autónoma o la DGX-A100 previamente descrita en el apartado 1.2.

Adicionalmente estos equipos cada vez cuentan con tecnologías más sotisficadas como la tecnología presentada por Nvidia en 2006 de la mano de Ian Buck llamada CUDA ,esta ofrece una solución a la computación a través de GPU , y la misma empresa la describía como ”una plataforma de computación paralela y un modelo de programación que permite incrementos dramáticos en el rendimiento de computación al aprovechar la potencia de la unidad de procesamiento de gráficos (GPU)”.

Dicha tecnología proporciona todo lo necesario para la aceleración de aplicaciones por GPU incluyendo entre otras cosas herramientas de desarrollo , librerías necesarias así como un compilador y es usada en campos muy diversos que requieren de grandes números de cálculos en paralelo , entre estos campos se puede encontrar la química computacional , el *machine learning* , la ciencia de datos , la computación dinámica de fluidos así como el campo de climatología.

Una forma de optimizar los tiempos de procesamiento de dichas aplicaciones es mediante la unificación de las GPU agrupando varias maquinas en lo que se conoce como *clusters* los cuales actualmente hay empresas que los ponen a disposición a través de servicios web de computación en la nube, ejemplo de ello son AWS (Amazon Web Services), Google Cloud o Azure. Estas 3 opciones están respaldadas por Amazon , Google y Microsoft respectivamente, aunque compiten en el mismo mercado tratando de ofrecer el mejor servicio dependiendo de las necesidades se puede destacar uno u otro ya que presentan pequeñas diferencias.

En el ámbito de la latencia AWS consigue ofrecer tiempos de 40 microsegundos aproximadamente siendo este el menor tiempo, por otra parte en el rendimiento de lectura/escritura de almacenamiento Azure destaca con 23300 iops (entradas salidas por segundo), por ultimo el servicio de Google consigue ofrecer velocidades de 25000 Mbps aproximadamente en *throughput* (tasa de transferencia efectiva) llegando a triplicar a sus rivales [11].

Por último cabe destacar que dichos *clusters*, orientados a la computación de altas prestaciones, cuentan con un sistema dedicado a la gestión de colas los cuales ofrecen grandes ventajas tanto para el usuario como para los nodos del *cluster*, ejemplos de dichos sistemas pueden ser Sun Grid Engine (SGE), PBS, Torque y Slurm entre otros.

Estos protocolos son bastante similares entre si por lo que no se presenta una clara ventaja de uno sobre otro, también se puede observar que presentan sintaxis muy similares, este es el caso de SGE y Slurm los cuales guardan cierta similitud especialmente a la hora de especificar los recursos necesarios para las tareas [12].

1.4. Objetivos del trabajo

Con el siguiente TFG se pretende realizar una correcta implementación de un protocolo automático para la gestión inteligente de tareas, con dicho protocolo y después de especificarle los equipos que conforman nuestro *cluster* sera capaz de recibir tareas y destinarlas a los servidores especificados, también nos ofrece ventajas como el balanceo automático de cargas entre colas para evitar sobrecargas.

Aprovechando la instalación de dicho protocolo también se realizará el despliegue de servicios de monitorización, con esto se consigue que desde una misma pantalla se puedan visualizar diferentes variables de interés , información como puede ser el estado de las tareas enviadas, el estado de los servidores de computo, etc hasta información especifica de cada servidor como puede ser la temperatura o potencia de las GPU entre otras variables.

De forma conjunta también se instalará un servicio el cual nos proporcionará una interfaz gráfica para el envío de tareas, ya que de esta forma se podrá realizar dicho trabajo de forma mucho más simple haciendo que el uso del protocolo de gestión de tareas sea mucho más intuitivo.

Por último se adentrará en el campo de la inteligencia artificial, generando una tarea básica que consiste en el reconocimiento de una serie de imágenes después de entrenar la inteligencia artificial durante un numero determinado de etapas , de esta forma se podrá evaluar el correcto funcionamiento del protocolo implementado simulando una situación cotidiana de uso.

Para conseguir realizar de forma exitosa se deberá:

- Explorar e investigar las principales herramientas para la gestión eficiente de colas de tareas del estado del arte.
- Definir la estructura del protocolo de gestión inteligente de tareas de inteligencia artificial y llevar a cabo la implementación del algoritmo.
- Explorar y analizar las principales herramientas de monitorización de tareas y estado de los servidores que forman parte del *cluster*. Implementación de este servicio.
- Explorar y analizar las principales herramientas para el envío de tareas mediante interfaz gráfica. Implementación de este servicio.
- Diseño e implementación de tareas propias de la inteligencia artificial.
- Despliegue de la solución definitiva en servidores de computación pertenecientes a un entorno real de inteligencia artificial.

1.5. Planificación del trabajo

A la hora de planificar el trabajo a desarrollar en este TFG nos hemos apoyado en los conocidos *Diagramas de Gantt* detectando cada uno de los pasos a realizar para llevar a cabo el despliegue y ajustando un tiempo para dicha tarea , de esta forma obtenemos un diagrama como el que se puede observar en la figura 1.1.

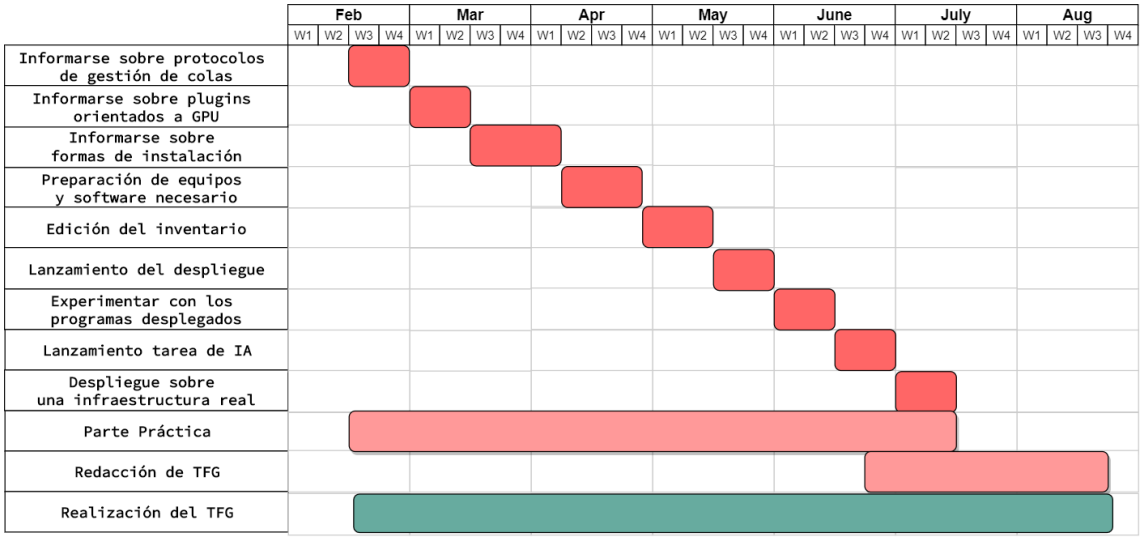

Figura 1.1: Planificación del trabajo

Capítulo 2

Marco teórico

2.1. Que es la inteligencia artificial

Entendemos por inteligencia artificial (IA) a todo conjunto de técnicas con las cuales se pretende que máquinas imiten el comportamiento humano, consiguiendo así que funcionen de manera autónoma, pudiendo aprender y adaptarse a diferentes situaciones y entornos. Esta rama de la informática ha tenido un crecimiento exponencial desde mediados del siglo pasado, esto es gracias en gran parte a un hardware cada vez mas potente y con mayor capacidad de computo que permite la ejecución de algoritmos altamente sotisficados, haciendo posible el análisis de grandes cantidades de datos.

Para hablar de los orígenes de la inteligencia artificial nos tendríamos que remontar a 1950 donde se pueden encontrar las primeras publicaciones, entre ellas publicación realizada por *Alan Turing* , considerado el padre de la IA así como de la informática , llamada *"Computing Machinery and Intelligence"* [13] en el cual se abordaba la siguiente pregunta "¿Pueden Pensar las máquinas?". En esta publicación se llevó a cabo la llamada *Prueba de Turing*, la cual estaba pensada para comprobar la eficacia de la inteligencia artificial. Esta prueba consiste en un examinador manteniendo una conversación escrita, mediante un ordenador, con una persona y con una maquina, la cual estaría programada para dar respuestas similares a las que darían los humanos, sin saber cual de los dos es la maquina o la persona, de esta forma si el examinador no consiguiese distinguir cual de los dos es la maquina, esta habría pasado la prueba correctamente.

Hoy en día la inteligencia artificial tiene infinidad de aplicaciones, muchas de ellas empleadas diariamente por nosotros aunque no seamos conscientes de ello, de las cuales se pueden destacar:

- Sector Comercial: con el cual consiguen ofrecer de forma mucho más personalizada productos por los cuales estarían interesados , esta técnica también es ampliamente usada en el sector del entretenimiento mediante plataformas como *Youtube* o Netflix el cual hace uso de la IA para recomendarnos cierto contenido basado en un conjunto de datos como pueden ser nuestras búsquedas y visualizaciones anteriores.
- Sector Salud: En el cual, mediante el uso de técnicas de IA, así como de imágenes biomédicas o señales fisiológicas, es posible la creación de herramientas capaces de asistir a especialistas en la toma de decisiones. La detección de enfermedades, tales como el Covid-19 o el Alzheimer [5]. Crear modelos predictivos analizando posibles patrones, gracias a muestras

genómicas y químicas. Como también, detectar automáticamente prótesis defectuosas en lineas de producción. [14]

- Sector Económico: En el sector económico se pueden encontrar aplicaciones interesantes tales como la detección de fraude mediante el análisis en tiempo real de una gran cantidad de transacciones y búsquedas de patrones de comportamiento que sean considerados como sospechosos, también se está usando en otro sector económico como es el de la bolsa con la cual se intenta predecir la posible fluctuación a la cual esta se vea sometida.
- Sector Seguridad: Este sector también se está viendo beneficiado por la IA ya que se utiliza para que de forma autónoma sea capaz de detectar problemas potencialmente peligrosos como podrían ser puertas traseras en una red y de esta forma obtener un análisis de posibles puntos a reforzar.
- Conducción Autónoma: Con el cual diversas empresas, relacionadas con el sector del automovilismo, hacen uso de la IA tratando de implementar tecnologías que permitan la conducción autónoma. El claro más visible es el que nos ofrece la empresa *Tesla* con su tecnología *autopilot* [15].

Dentro de la IA se pueden encontrar dos subcapas las cuales se han ido desarollando desde 1980 llamadas *Machine Learning* y *DeepLearning* quedando el siguiente modelo:

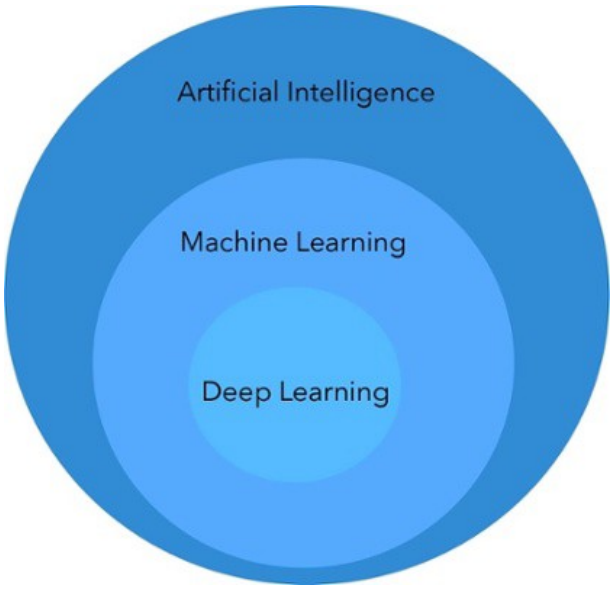

Figura 2.1: Capas de la IA

El *Machine Learning* consiste en dotar a las máquinas de aprendizaje automático usando tres tipos de aprendizaje ; aprendizaje supervisado , no supervisado y reforzado. Estas tres técnicas son tres formas diferentes de monitorizar y premiar el correcto aprendizaje de la IA descartando así formas erróneas, para saber que aprendizajes se están llevando a cabo de forma correcta o no, se hace uso del *ground truth*.[1]

[1] modelos creados de forma manual los cuales son usados como referencia para comparar los resultados del aprendizaje

Según como ha ido evolucionando el *Machine Learning* han ido surgiendo técnicas como el *clustering*, diferentes tipos de modelos para la resolución de un amplio abanico de problemas [16], redes neuronales, así como el aprendizaje profundo, más conocido como *Deep Learning*.

Este se basa en añadir un mayor número de capas a las redes neuronales, siendo estas cada vez mas profundas. Con esto, junto con un mayor número de datos de entrada a los que se utilizarían en *Machine Learning*, las redes neuronales consiguen aprender de forma totalmente autónoma siendo capaces de extraer características clave de los datos proporcionados por si mismas.

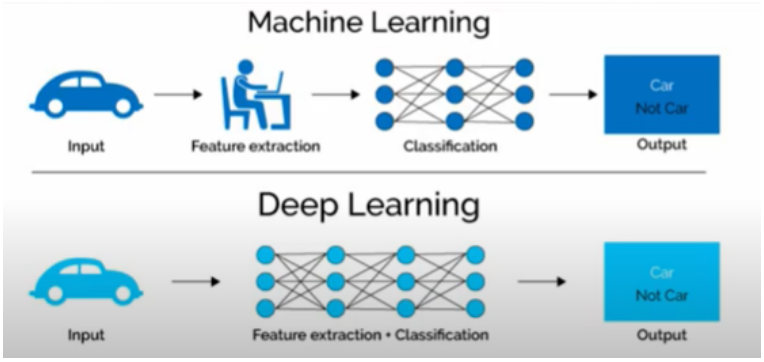

Figura 2.2: Comparación entre MachineLearning y DeepLearning

2.2. Como se entrena la inteligencia artificial

Las redes neuronales cuentan con una primera capa de entrada, una capa de salida, y en la mayor parte de los casos una o más capas intermedias, estas reciben el nombre de capas ocultas. Una representación de estas redes se muestra en la figura 2.3.

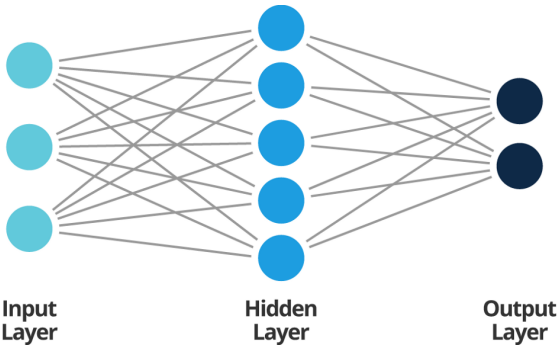

Figura 2.3: Capas de red neuronal artificial

Cada una de estas capas cuentan con una serie de neuronas, las cuales han sido representadas en la figura 2.3 como círculos. Además cuentan con diversos enlaces que interconectan neuronas de diferentes capas, además tienen asignado un peso para cada uno de estos enlaces, añadiendo así diferentes probabilidades de que se escoja un enlace u otro, y consecuentemente una neurona u otra. Estos pesos se modifican de forma autónoma e iterada buscando así decrementar lo máximo posible la función de perdidas de la red neuronal. Este proceso se puede observar en la gráfica 2.4

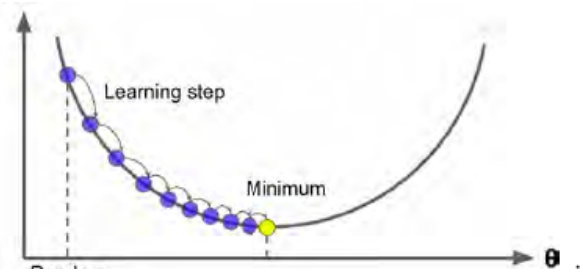

Figura 2.4: Optimización función de pérdidas

Entrando más en detalle, se conoce como descenso del gradiente, el cual busca mediante el uso del gradiente y derivadas aproximarse de forma iterada al mínimo absoluto. Si el avance de dichas iteraciones es demasiado grande puede resultar en que no converja, saltando de forma repetida e indefinida de un lado de la parábola, con derivada negativa, al lado contrario, con derivada positiva.

Para realizar el cálculo de dicho gradiente, las redes neuronales hacen uso de un algoritmo llamado *back-propagation*, con el cual se calcula el error una vez obtenida la salida, para seguidamente, hacer una propagación hacia atrás indicando a cada una de las neuronas su porcentaje de error, de esta forma estas son capaces de ajustarse cada vez más para optimizar dicho error final.

Las redes neuronales utilizadas en la inteligencia artificial han ido incrementando su complejidad con el paso del tiempo pasando a diseños más sotisficados entre las que destacaremos la red neuronal convolucional (CNN), la cual es utilizada, entre otras cosas, para tareas de visión artificial como la clasificación de imágenes o la detección de objetos. Sin entrar en gran detalle se puede observar que la topología de este tipo de red es la mostrada en la Figura 2.5.

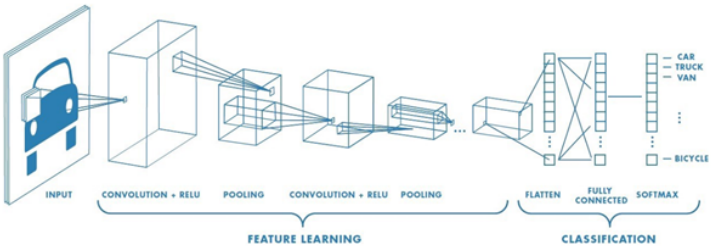

Figura 2.5: Topología de la red neuronal de una CNN

Para que estas redes puedan realizar tareas como las descritas anteriormente es necesario el análisis de una gran cantidad de datos. Esto incide directamente en la arquitectura del modelo haciendo que incremente su complejidad al tener que introducir más neuronas (unidad básica), estas necesitan de una gran potencia computacional para poder procesar los datos, ya que cada una de ellas se encargará de realizar una serie de cálculos sobre un píxel en particular, en el caso que los datos de entrada sean imágenes, realizándose así tantos cálculos en paralelo como píxeles tenga la imagen. Los cálculos que se realizan dentro de una CNN dependen de en que fase del proceso se encuentre, ya que este se subdivide en dos: fase de entrenamiento y fase de inferencia.

Entrando en detalle en la fase de entrenamiento destacaremos dos capas las cuales se describen a continuación.

La capa de convolución se encarga de realizar productos escalares de una matriz llamada kernel con grupos de píxeles contiguos de la imagen entrante, dicho proceso termina cuando se han procesado todos los píxeles de dicha imagen. Como se puede visualizar en la Figura 2.6.

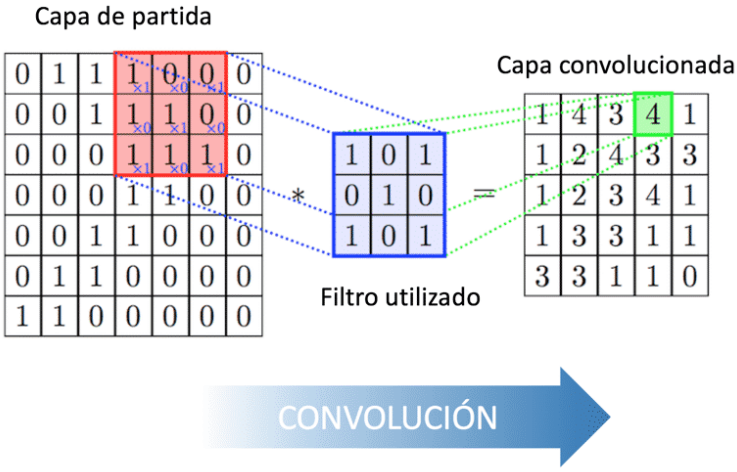

Figura 2.6: Convolución de matriz de entrada con filtro (Kernel)

La capa de *pooling* busca reducir la información a analizar preservando aquellas características de interés, de esta forma se disminuyen el numero de neuronas necesarias en las posteriores capas, evitando así el coste computacional preservando toda la información relevante. En la Figura 2.7 se puede observar como se mantienen aquellos valores más altos de cada uno de los grupos de la matriz.

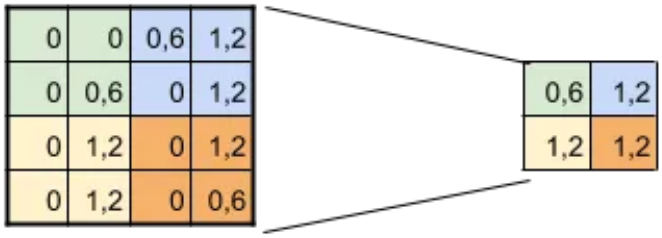

Figura 2.7: Proceso de pooling sobre una matriz dada

Centrándonos a continuación en las aplicaciones de las CNN se puede destacar el uso de estas para el procesamiento de imágenes, las cuales cada vez cuentan con mayor resolución, esto implica mayor numero de cálculos en paralelo entre otras cosas, es por razones como esta que la mejor opción es usar GPU como procesador encargado de ejecutar tareas de IA, las cuales han demostrado ser la mejor opción a la hora de ejecutar gran cantidad de cálculos de forma simultanea y paralelo.

¿Pero por que es mejor opción utilizar GPU a las CPU ? para ello se debe tener en cuenta la función que desempeñan cada uno de estos dos tipos de procesadores, las llamadas CPU son unidades de procesamiento con propósito general, las cuales tienen que ser capaces de poder procesar

diferentes tipos de tareas. Las GPU son procesadores cuyo propósito es más especifico y este está orientado a la ejecución en paralelo de tareas, es por esto que cuentan con mayor numero de núcleos, lo cual es altamente beneficioso para las redes neuronales, ya que en estas se busca la ejecución simultanea de cálculos sencillos en cada una de sus neuronas.

En una publicación realizada en 2008 llamada “Large-scale deep unsupervised learning using graphics processors“ [8] sostiene que las GPU son capaces de ejecutar tareas de *deep learning* 70 veces más rápido que mediante el uso de CPU multinucleo.

2.3. Qué es un cluster

La palabra *cluster*. proveniente del inglés cuyo significado es grupo, aunque esta pueda ser utilizada en diferentes campos, también se utiliza en el ámbito tecnológico para hacer alusión a una agrupación de ordenadores cuyo objetivo es igual o muy similar, de esta forma dichas agrupaciones se pueden ver como una única máquina, cada una de estas máquinas recibe el nombre de nodos o *Slaves* y suelen estar coordinados por un nodo central o *Master*, el cual controlará y dirigirá el trabajo a realizar a lo largo de los nodos del *cluster*.

Los nodos pueden agruparse en particiones y pueden encontrarse en mas de un grupo simultáneamente, si el administrador así lo ve necesario. La finalidad de estas agrupaciones es básicamente distinguir a unos grupos de otros dependiendo de:

- Las diferentes capacidades de cómputo, ofreciendo así la posibilidad de utilizar una partición u otra en base a los requerimientos, mas o menos exigentes, de cada tarea.
- Las tareas a las que se van a dedicar, pudiendo tener una partición que se encargue enteramente a una serie de tareas y adicionalmente contar con otra partición que procese otro tipo diferente de tareas.

Dentro de los *clusters* se pueden encontrar diferentes tipos dependiendo de la carga de trabajo que tengan que procesar, su disponibilidad, etc. Es por ello que estos se pueden agrupar en:

- High Performance Computing (HPC) Estos *clusters* están dedicados al computo de grandes cantidades de información , son los indicados para el entrenamiento de algoritmos de inteligencia artificial ya que se necesita de la ejecución en paralelo sobre varios procesadores de forma simultanea. Este tipo de *clusters* es utilizado por Amazon en su servicio de computación en la nube *Amazon Web Services (AWS)* o por Microsoft en su servicio *Azure*.
- High Performance Service (HPS) Este tipo de clusters están orientados a poder atender gran numero de peticiones de diferentes usuarios como seria el caso de servicios web.

Centrándonos en los HPC, debemos destacar que son *clusters* de GPU’s los cuales están orientados a conseguir una velocidad muy alta de cálculo y un gran número de operaciones en paralelo, característica muy interesante para el calculo de bloques en tecnologías de *block-chain*, por ello se usan *clusters* en granjas de criptomonedas ya que se necesita realizar cálculos matemáticos complejos de forma iterada o incluso para entrenamiento de inteligencia artificial entre otras tareas, es por ello que cada uno de los nodos pertenecientes al *cluster* cuentan con sus pertinentes GPU siendo estas la pieza clave de dicha agrupación.

2.4. Gestor de colas

Para entender la necesidad de *software* encargado de la gestión de colas nos centraremos en las opciones que disponemos a la hora de enviar tareas/*scrips* al *cluster* en cuestión.

La forma mas sencilla de realizarlo seria mediante una conexión remota utilizando SSH , esto permite que desde el terminal puedas ejecutar las tareas pertinentes en el nodo deseado. Como se puede imaginar, esta opción desaprovecha el gran potencial que nos ofrece el *cluster* ya que continuamos viendo los nodos como máquinas independientes y realizamos manualmente el envío y ejecución de las tareas , es por eso que debemos pensar más en el nivel de abstracción que nos ofrece el tener todos los nodos agrupados en lo que llamamos *cluster* y también en los diversos subgrupos que se puedan generar dentro de este.

Este hecho es el que motiva la existencia de algoritmos que simplifiquen dicho trabajo como puede ser el caso de la librería *Slurm*, que ofrece la posibilidad de encolar tareas a un grupo de nodos de computación (particiones) o también a un nodo en concreto , todo desde un mismo ordenador al cual se le conoce como master. Dicha distribución la se puede observar en la figura 2.8

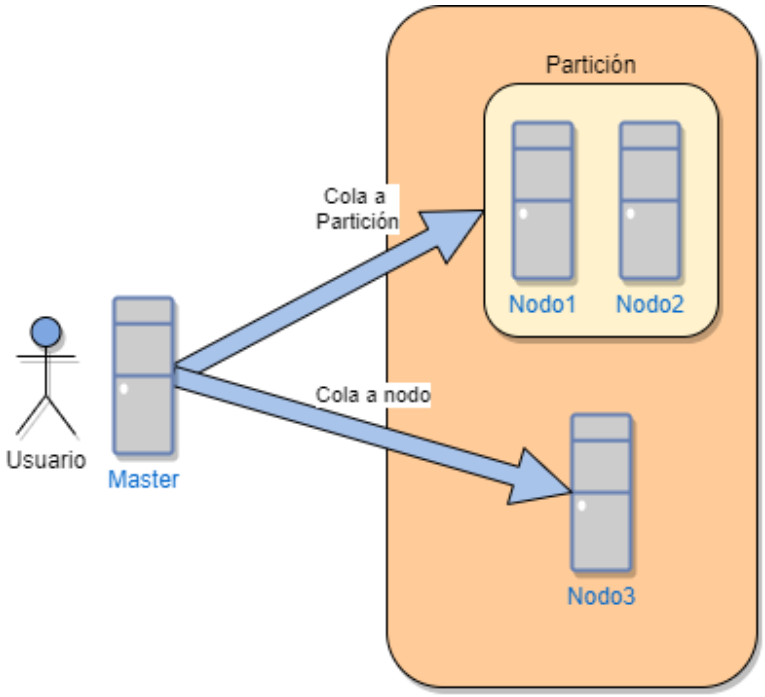

Figura 2.8: Diagrama de master , nodos y particiones

Capítulo 3

Material empleado

Para realizar el despliegue del algoritmo de gestión de colas que se pretende implementar, así como para llevar a cabo una monitorización en tiempo real del estado del *cluster* y poder enviarle tareas mediante interfaz gráfica, es necesario describir diversas tecnologías que posibiliten todas estas funcionalidades. A lo largo de este capítulo se detallan las principales particularidades de las herramientas empleadas para el desarrollo del presente Trabajo Fin de Grado.

3.1. Slurm

Slurm, cuyo logo se muestra en la Figura 3.1, es un protocolo de gestión de colas, el cual permite a sus usuarios el envío de tareas hacia los nodos de computación de un *cluster*.

Figura 3.1: Logo de Slurm

Dicho protocolo no cuenta con una interfaz gráfica, es por esto que tanto el envío de tareas como la consulta de información de interés se realiza mediante el uso de una serie de comando a través del terminal. Entre los comando que esta herramientas posibilita utilizar, destacaremos los siguientes:

- *sinfo* nos muestra información referida al estado de todas las particiones como puede ser sus

nombres, disponibilidad de cada una de las particiones o el numero y nombre de los nodos que conforman cada agrupación.

- *squeue*: este comando está relacionado con el estado de las tareas, nos muestra un listado de aquellas tareas que no han sido finalizadas junto con su ID, el tiempo que ha pasado desde su envío, el nodo el cual está procesando o procesará dicha tarea o su estado actual, ya que una tarea puede encontrarse en ejecución, en espera a ser atendida o descartada si ha habido algún error.

- *scancel*: la función de este comando radica en cancelar el envío o la ejecución de una tarea en especifico, para ello se le deberá facilitar al programa el ID de la tarea que se desea interrumpir, dicho ID se puede obtener mediante el comando previamente explicado 3.1

- *sbatch*: encargado del envío de las tareas a los nodos de cómputo, su funcionamiento radica en especificar seguidamente la ruta del archivo a enviar el cual se debe encontrar en formato *batch*, el cual consiste en en la especificación de una serie de instrucciones por lotes.

A la hora de hacer uso del comando *sbatch* contamos también con diversas opciones adicionales, las cuales nos pueden ayudar a especificar los recursos que queremos emplear en dicha tarea. Para ello se escribirá a continuación del comando *sbatch* y antes de la ruta de la tarea, tantos parámetros como especificaciones queramos darle. A continuación se enunciaran las más utilizadas.

- *–ntasks= <numero>*: permite especificar el numero de CPU que procesarán dicha tarea.

- *–gres=gpu:<numero>*: podremos elegir el número de GPU que ejecutarán la tarea.

- *–mem= <numero con unidades de memoria>*: destinado a determinado la memoria máxima que podrá ser usada para dicha tarea.

- *–time=<hh>:<mm>:<ss>*: utilizado a especificar el tiempo máximo que dicha tarea puede encontrarse en ejecución.

- *–nodes= <numero>*: especifica el número de nodos de computación que deberán ejecutar dicha tarea.

A la hora de configurar *Slurm* este se basta de un documento de texto llamado *slurm.conf* [17], en este deberemos especificar las IP de cada uno de los nodos de cómputo así como los grupos de nodos que queramos realizar (particiones), entrando en detalle también se nos ofrece la posibilidad de designar como predeterminado una partición en concreto, asignar el tiempo máximo de espera por defecto de las tareas en cola, limitar la memoria máxima que cada una de estas colas puede consumir, si en el tarea no se especifica un valor diferente, o elegir el tipo de planificador a utilizar en cada una de las colas, pudiendo elegir así entre el modelo FIFO (firts in , firts out) o *Backfill* [18], el cual tiene en cuenta las necesidades computacionales de cada una de las tareas , consiguiendo mayor optimización en el tiempo como se puede observar en la imagen 3.2 de forma muy gráfica.

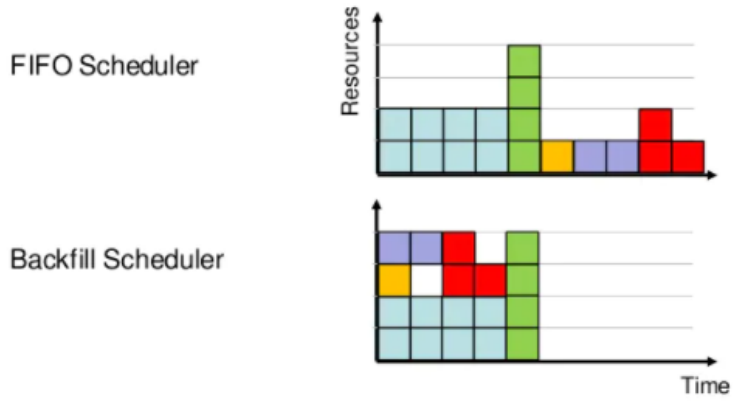

Figura 3.2: Comparación entre planificador FIFO y Backfill

3.2. Ansible

Se trata de una plataforma de software libre, cuyo propósito principal es centralizar el despliegue de la configuración y administración de un *cluster*, de esta forma dicho despliegue se realiza desde un servidor que hará las veces de controlador hacia el resto de equipos del *cluster* que reciben el nombre de nodos. El logo de dicho programa se puede visualizar en la Figura 3.3.

Figura 3.3: Logo de Ansible

Entre sus características destaca el tratarse de un programa consistente con un alto grado de seguridad, ya que no instala ningún tipo de agentes requiriendo únicamente una conexión *SSH* con los nodos, siendo esta forma altamente comprobada y poco vulnerable. También su suave curva de aprendizaje ya que gran parte del funcionamiento de este programa se basa en la ejecución de documentos llamados *playbooks*, los cuales están escritos en *YAML*. Este lenguaje de marcado descriptivo es utilizado para mapear cualquier tipo de dato de lenguajes de alto nivel, con una notación basada en la sangría que resulta altamente sencilla de comprender. En la Figura 3.4 se puede observar la sintaxis de dicho lenguaje [19].

```yaml
- hosts: all
  become: yes
  roles:
    - name: facts

# Set root SSH key to allow SSH/ansible when no slurm job running for ansible user
- hosts: slurm-node
  become: yes
  tasks:
    - name: Add SSH public key to root user authorized keys
      authorized_key:
        user: root
        state: present
        key: "{{ lookup('file', ansible_ssh_private_key_file | default(lookup('env','HOME') + '/.ssh/id_rsa') + '.pub') }}"

# Build slurm first on all nodes
- hosts: slurm-cluster
  become: yes
  tasks:
    - name: build slurm on all nodes
      include_role:
        name: slurm
        tasks_from: build
```

Figura 3.4: Sintaxis de YAML

3.2.1. Inventario

En primer lugar, Ansible cuenta con un documento de texto, denominado inventario, en el cual se debe especificar información crucial del *cluster* sobre el que se llevará a cabo el despliegue. En dicho documento encontraremos entre otros las IP de los nodos, las rutas a los archivos que contienen las claves *SSH* de las maquinas, las cuales suelen ubicarse dentro de la carpeta .ssh, así como la especificación de que rol va a asumir cada uno de los nodos en caso de desplegar programas que tengan una relación *Master-Slave* entre ellos.

Como se puede observar en la Figura 3.5, primeramente se especifican los nodos que van a conformar el *cluster* junto con datos de interés los cuales se utilizarán durante el despliegue, posteriormente también se especifica la relación que los equipos tendrán entre ellos en el programa de gestión de colas.

```
######
# ALL NODES
# NOTE: Use existing hostnames here, DeepOps will configure server hostnames to match these values
######

[all]

master-slurm    ansible_host=000.00.000.000  ansible_ssh_user=xxxx ansible_ssh_pass=xxxx ansible_ssh_private_key_file=/home/
xxx/.ssh/<nombre_archivo>   ansible_sudo_pass=xxxxx

cvblab03 ansible_host=000.00.000.00 ansible_port=xx ansible_ssh_user=xxxx ansible_sudo_pass=xxxx ansible_ssh_private_key_file=/home/
xxx/.ssh/<nombre_archivo> ansible_ssh_pass=xxxx

cvblab04 ansible_host=000.00.000.00 ansible_port=xx ansible_ssh_user=xxxx ansible_sudo_pass=xxxx ansible_ssh_private_key_file=/home/
xxx/.ssh/<nombre_archivo> ansible_ssh_pass=xxxx

######
# SLURM
######
[slurm-master]
master-slurm

[slurm-node]
cvblab03
cvblab04

[slurm-cluster:children]
slurm-master
slurm-node
```

Figura 3.5: Extracto de un inventario de Ansible

3.2.2. Playbooks

Son archivos escritos en *YAML* encargados de describir/dirigir el despliegue o configuración que se desea realizar sobre los nodos del *cluster*.

Estos pueden hacer alusión a un grupo en específico de nodos los cuales vaya a tener un mismo rol. En dichos *playbooks* encontramos el uso de tareas las cuales se irán ejecutando de forma secuencial sobre los nodos para completar así el despliegue/configuración.

En la Figura 3.4 se puede observar parte de un *playbook* el cual se ejecutará a la hora de lanzar el despliegue.

3.2.3. Módulos

Son las unidades básicas de trabajo, las cuales se escriben para ser modelos de los estados que se desean alcanzar en dichos sistemas. Para ello se utilizan leguajes descriptivos tales como *Python*, *Bash* o *Ruby* entre otros y se ejecutan mediante una conexión SSH a los equipos a configurar.

Estos módulos, los cuales forman parte de los *Playbooks*, están programados de forma que se puedan ejecutar, reiteradas veces consiguiendo siempre el mismo resultado y permitiendo así ejecutar de nuevo un *Playbook* el cual no haya finalizado de forma satisfactoria, por lo tanto, dicha ejecución se realizará a partir del último módulo correctamente finalizado.

3.3. Docker

Docker es un software libre, cuyo logo se puede visualizar en la Figura 3.6. Dicho servicio es utilizado para empaquetar programas en estancias llamadas contenedores, los cuales cuentan con todo lo necesario para la ejecución del software como: bibliotecas, código e incluso los recursos que se invertirán en dicho contenedor.

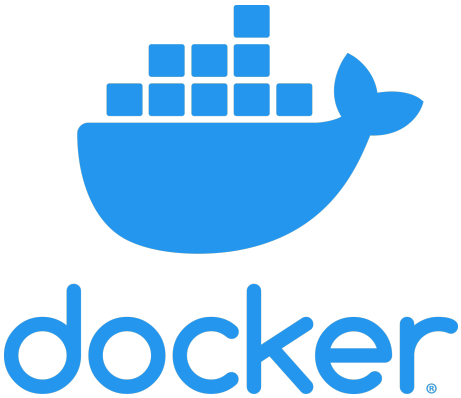

Figura 3.6: Logo de Docker

Este programa hace uso de mecanismos de aislamiento, propios de *linux*, como pueden ser los *cgroups* o *namespaces*, permitiendo la ejecución en paralelo de múltiples programas en sus correspondientes contenedores y evitando así problemas de incompatibilidad de versiones.

Jay Liman, analista de la industria 451 Research afirma que, ”Docker es una herramienta que puede empaquetar una aplicación y sus dependencias en un contenedor virtual que se puede ejecutar en cualquier servidor Linux. Esto ayuda a permitir la flexibilidad y portabilidad en donde la aplicación se puede ejecutar, ya sea en las instalaciones físicas, la nube pública, nube privada, etc.”[20].

3.4. Github

Se trata de una plataforma web, cuyo logo se muestra en la Figura 3.6, destinada al control de versiones de proyectos con opción a proyectos colaborativos, en la que se puede trabajar de forma independiente en líneas paralelos de edición, así como ofrecer la posibilidad de restaurar versiones anteriores de dicho proyecto [21]. También se puede contar con *Git*, el cual es la versión de escritorio de *Github*, la cual te da la opción de trabajar con una terminal de comandos o con un entorno gráfico para sincronizar los directorios sobre los que se está trabajando.

Figura 3.7: Logo de GitHub

En la plataforma *Github* se pueden encontrar repositorios públicos ajenos que se pueden rápidamente descargar o clonar a nuestro escritorio.

3.5. Prometheus

Prometheus es un servicio de monitorización para los nodos de un *cluster*, cuyo logotipo se puede visualizar en la Figura 3.8, el cual registra métricas en tiempo real y las almacena en una base de datos para ser consultadas por el usuario. Adicionalmente permite crear alarmas si ciertos estados de los equipos superan un umbral a establecer, esto es de gran ayuda para evitar posibles daños en los equipos por razones como una temperatura excesivamente alta durante un tiempo prolongado.

A la hora de realizar dichas consultas se debe acceder al servicio web que ha levantado *Prometheus*, para posteriormente poder visualizar mediante gráficas temporales la temperatura del equipo, el uso de memoria, porcentaje de potencia empleada, etc. Cabe destacar que la consulta de dichas variables se hace mediante una barra de búsqueda y el uso de nombres no tan intuitivos lo cual dificulta un poco la experiencia de usuario.

Figura 3.8: Logo de Prometheus

En la Figura 3.9 se detalla de forma esquemática la arquitectura de dicho servicio entre el que destacan: el servidor Web el cual se ha explicado anteriormente, su sistema de alertas que permite notificar por diferentes medios al usuario o por último la posibilidad de exportar datos mediante PromQL a otros servicios [22].

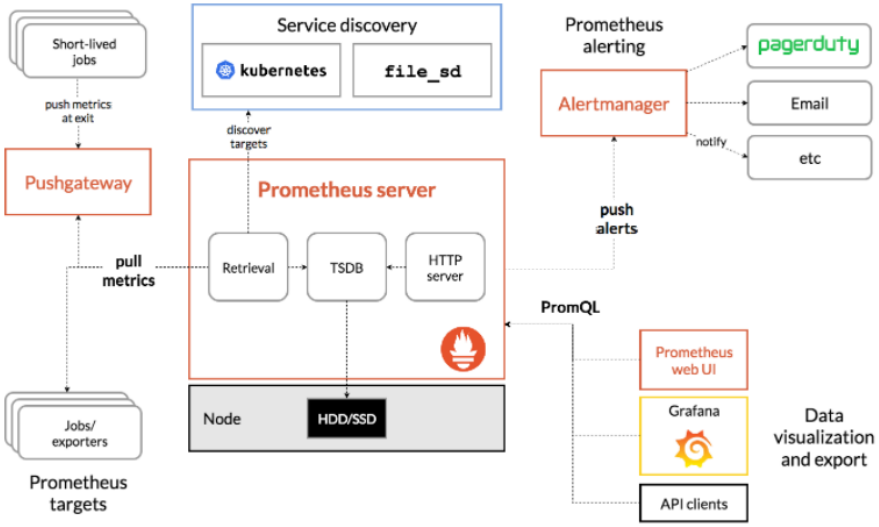

Figura 3.9: Esquema de la arquitectura

3.6. Grafana

Grafana,cuyo logo se visualiza en la Figura 3.10,también es un servicio de monitorización de los nodos de un *cluster* que ofrece información del estado de los nodos en tiempo real. *Grafana* ofrece una interfaz mas limpia que *Prometheus* con una amplia variedad de variables mostrándose de forma simultanea, además de diferentes utilidades. Se puede considerar que añade una nueva capa de abstracción a la consultas anteriormente descritas.

Figura 3.10: Logo de Grafana

Dicho *framework* se puede desplegar de forma conjunta con *Prometheus*, este actuará como la base de datos a la que accederá *Grafana* para poder representar mediante *dashboards* el estado de los nodos, dicho proceso se realiza mediante consultas PromQL y se puede visualizar en la Figura 3.9. La información a visualizar es la misma a la que se puede visualizar en *Prometheus* aunque en este caso, se presenta de forma mas limpia e intuitiva lo cual optimiza la experiencia de usuario.

Dichos *dashboards* muestran una gran variedad de información de forma simultanea de un nodo del *cluster* en concreto mediante la representación de gráficas, pudiendo seleccionar que nodo se quiere visualizar mediante un desplegable, el cual está ubicado en la parte superior del tablero, así como la base de datos la cual por defecto solo se dispondrá de *Prometheus*.

3.7. OpenOnDemand

Se trata de una servicio web, logotipo el cual se puede visualizar en la Figura 3.11, el cual ofrece una interfaz gráfica orientada al envió de tareas a un *cluster* de computación, en este caso haciendo uso del sistema de gestión de colas llamado *Slurm*. Esta plataforma es accesible vía http, escribiendo la IP del nodo *Master* junto a su numero de puerto que por defecto es 9050.Este número de puerto, puede ser cambiado modificando una variable en el *Playbook* que *Ansible* utilizará para realizar el despliegue, por lo que el cambio hay que realizarlo previo a este.También ofrece un servicio de autenticación por lo al tratar de acceder a dicha plataforma se requerirá de usuario y contraseña.

Figura 3.11: Logo de OpenOnDemand

En este servicio se puede encontrar la opción de *Job Composer*, la cual se muestra en la Figura 3.12, esta nos ofrecerá la opción de crear nuevos *Jobs* a partir de una plantilla por defecto , de plantillas previamente modificadas o de *jobs* previamente enviados.También permite visualizar

toda la información referente a cada uno de los *jobs* previamente enviados, esto posibilita revisar desde el código enviado como los posibles *outputs* que se hayan obtenido después de procesar el *script*.

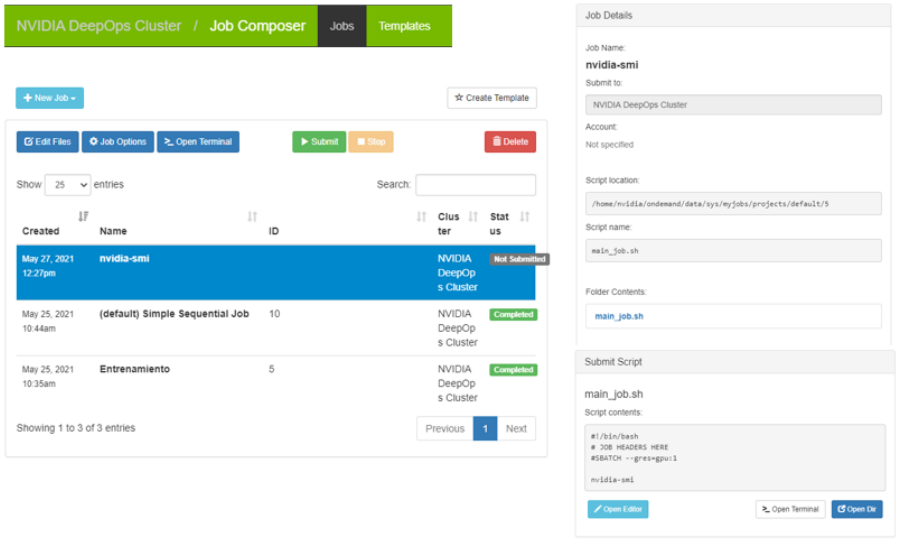

Figura 3.12: Job Composer

3.8. Servidores

En el caso del presente trabajo fin de grado, se utilizarán tres servidores que es el numero mínimo necesario para poder realizar el despliegue y poner en marcha todas las funcionalidades que se han ido presentando en el capítulo actual. Cada uno de los servidores tiene un rol diferente el cual se puede observar en la Figura 3.13 y se detalla a continuación:

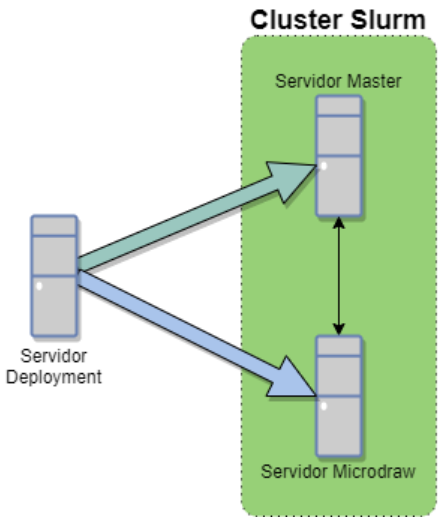

Figura 3.13: Representación esquematizada de los roles de cada servidor

3.8.1. Servidor Deployment: Datathon

Como se puede observar en la Figura 3.13, este servidor está pensado para ubicarse fuera del *cluster* final, ya que será el encargado de realizar el despliegue de los diversos servicios sobre los nodos que conformen dicho *cluster*, para dicha tarea el servidor hará uso de *Ansible*.

3.8.2. Servidor Master

Este servidor será el encargado de asumir el rol de Master para *Slurm*, es decir, este será el nodo al cual se habrá que conectar, bien vía remota o bien físicamente, para poder encolar las tareas a un conjunto de nodos en concreto.

Sobre este servidor también se despliega el servicio web *OpenOnDemand*, descrito en el apartado 3.7, al cual se podrá acceder para mandar tareas contando con una interfaz gráfica.

3.8.3. Servidor Slave: MicroDraw

Será el encargado de asumir el rol de Slave para *Slurm* , es decir , este será el nodo encargado de realizar todas aquellas tareas enviadas desde el Master pasando a ser un nodo de computo del cluster.

Dado que es un nodo de computo, también contaremos con los programas *Grafana* y *Prometheus* para poder monitorizar este servidor, pudiendo visualizar información referida con el estado de las tareas así como información de las GPU que monta este servidor.

Capítulo 4

Desarrollo del proyecto

En este capítulo se abordará como realizar el despliegue de las herramientas previamente descritas. Para ello, principalmente se deberán configurar correctamente los equipos así como los programas previamente descritos, posteriormente se comprobará el correcto funcionamiento del software instalado pasando así a realizar diferentes mediciones. Por último se realizará el envío de una tarea de IA al *cluster*, la cual está basada en el reconocimiento de imágenes mediante el uso de una glscnn.

4.1. Despliegue de un sistema para la gestión de colas

Habilitar SSH

Uno de los primeros pasos es habilitar que se puedan realizar conexiones SSH en los 3 equipos que usaremos para el despliegue , para ello se escribirán las siguientes lineas en el terminal:

sudo apt update

sudo apt install openssh-server

Una forma de comprobar que este servicio se ha instalado y se está ejecutando de forma correcta es mediante el uso del comando *sudo systemctl status ssh*.

Una vez introducido dicho comando se mostrará la siguiente información 4.1 confirmando así que el proceso se ha realizado correctamente como se puede observar en la Figura 4.1.

```
nvidia@microdraw:~$ sudo systemctl status ssh
● ssh.service - OpenBSD Secure Shell server
     Loaded: loaded (/lib/systemd/system/ssh.service; enabled; vendor preset: enabled)
     Active: active (running) since Tue 2021-06-22 05:53:09 UTC; 4 weeks 0 days ago
       Docs: man:sshd(8)
             man:sshd_config(5)
   Main PID: 793 (sshd)
      Tasks: 1 (limit: 28741)
     Memory: 11.0M
     CGroup: /system.slice/ssh.service
             └─793 sshd: /usr/sbin/sshd -D [listener] 0 of 10-100 startups
```

Figura 4.1: Consulta del estado del servicio de SSH

Instalar Git

Necesitaremos instalar este software para poder clonar el repositorio que emplearemos para realizar el despliegue en nuestro *cluster*, para ello se introducirán los siguientes comandos en el terminal:

sudo apt update

sudo apt install git

A la hora de consultar su correcta instalación se puede hacer uso del comando *git –versión*.

Dicho comando nos mostrará la correspondiente versión que se ha instalado en nuestro equipo, como se muestra en la captura 4.2.

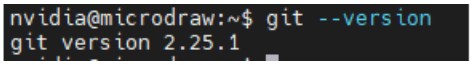

Figura 4.2: Consulta de la versión de *Git* actualmente instalada

Instalar Python

También debereremos instalar *python* el cual es necesario ya que *Ansible* necesita una versión superior a la 2.4 de este lenguaje de programación. Para ello se escribirá en el terminal los siguientes comandos:

sudo apt-get update

sudo apt-get install python

Una vez realizada la instalación, se podrá comprobar la versión instalada mediante el comando: *python –version* como se muestra en la figura 4.3.

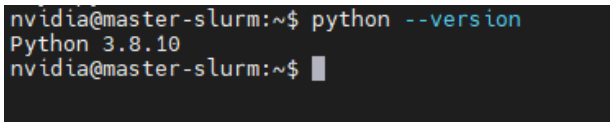

Figura 4.3: Consulta de la versión de *Python* actualmente instalada

Crear punto de restauración

Para realizar la siguiente tarea haremos uso del software *TimeShift* el cual nos ofrecerá la posibilidad de crear puntos de restauración a los cuales podremos regresar en caso de encontrarnos con algún error irreversible durante el despliegue.

Para su instalación deberemos escribir los siguientes comandos:

sudo add-apt-repository -y ppa:teejee2008/timeshift

sudo apt update

sudo apt install timeshift

Una vez realizada la instalación procederemos a crear el punto de restauración, en este caso se realizará por medio del terminal, lo cual permite realizarlo remotamente desde otro equipo si fuera necesario mediante la conexión SSH, previamente habilitada. Para ello introduciremos el siguiente comando:

sudo timeshift –create –comments "Primer punto de restauración-tags D

En cuanto el punto de restauración haya sido creado se visualizará a través de una lista, como la mostrada en la figura 4.4, mediante el comando:

sudo timeshift –list

```
nvidia@microdraw:~$ sudo timeshift --list

/dev/sda5 is mounted at: /run/timeshift/backup, options: rw,relatime,errors=remount-ro

Device : /dev/sda5
UUID   : e46e320d-b480-42cd-818b-1761a97c9479
Path   : /run/timeshift/backup
Mode   : RSYNC
Status : OK
2 snapshots, 51.9 GB free

Num     Name                 Tags  Description
------------------------------------------------------------------------------
0    >  2021-05-11_18-15-53  D     python ,claves ssh , drivers de nvidia y cuda instalado
```

Figura 4.4: Listado de puntos de restauración creados

Clonar Repositorio en Deployment

Una vez los equipos cuenten con las configuraciones necesarias, pasaremos a descargar el repositorio de *GitHub* el cual hará uso de *Ansible* para realizar el despliegue.

Para realizar el despliegue sobre nuestro *cluster* necesitaremos clonar el repositorio de *Nvidia Deepops* a nuestro ordenador. En este caso se clonará en el ordenador *"Datathon"* el cual se encargará de orquestar el despliegue.

Entrando en https://github.com/NVIDIA/deepops encontraremos la opción de copiar el *link* que nos permitirá el clonado desde nuestro escritorio como se muestra en la Figura 4.5.

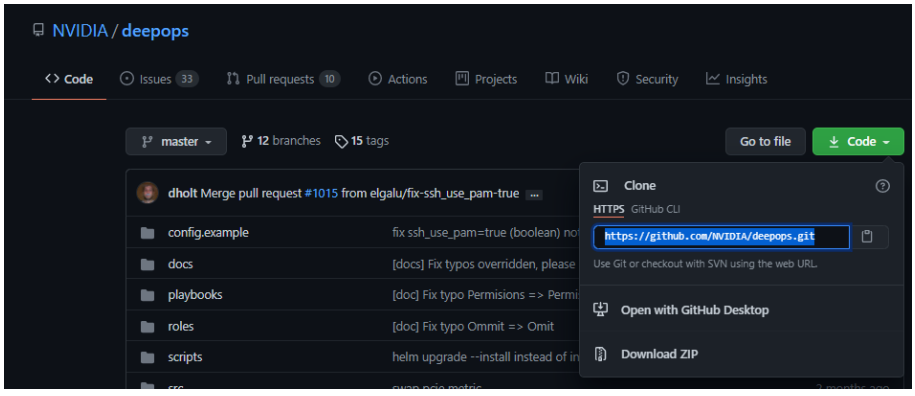

Figura 4.5: Procedimiento para obtener el *link* de clonado del repositorio

Una vez copiado el *link* accederemos en al terminar y haremos uso de la aplicación *Git* para realizar dicho clonado, para ello escribiremos el siguiente comando:

git clone https://github.com/NVIDIA/deepops.git

Este proceso tardará unos segundos, ya que tiene que realizar varias tareas antes de estar finalizado, durante esta espera, se nos mostrará el avance del clonado como se representa en la Figura 4.6.

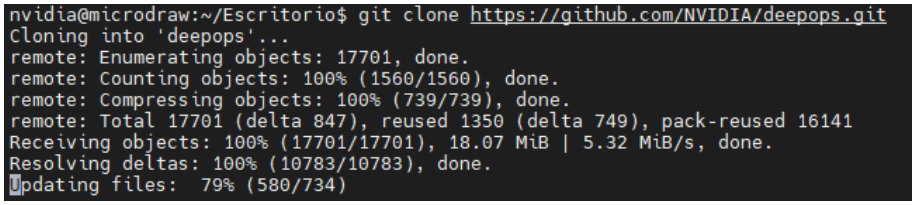

Figura 4.6: Proceso de clonado del repositorio *Nvidia DeepOps*

Una vez finalizado tendremos nuestra carpeta creada con todos los archivos del repositorio clonados en nuestro directorio de trabajo.

Instalar Ansible

Se accederá a la carpeta anteriormente clonada y se abrirá un terminal desde ahí para ejecutar el siguiente comando:

./scripts/setup.sh

Dicha instrucción instalará en nuestro equipo *Ansible* además de paquetes de *Python* y diversas utilidades más.

Dado que en los siguientes puntos se continuará haciendo uso del terminal ubicado en la carpeta se recomienda no cerrarlo.

Editar el inventario

Como se explicó en el apartado 3.2.1 uno de los archivos clave que forma parte de *Ansible* es el inventario. En dicho fichero, se almacena información pertinente al *cluster*. Por ello, en este punto se procederá a rellenar aquellos campos que son necesarios para el correcto despliegue de los diferentes *softwares* en el *cluster*.

En el inventario, generado por defecto, se puede observar como, en lineas comentadas mediante el carácter #, se muestra como se debe especificar la información requerida de cada equipo. Adicionalmente se deberá especificar ciertos campos, los cuales no se encuentran por defecto en el documento, como son:

- *ansible_ssh_user*:En este campo se especificará el nombre actual de cualquiera de los usuarios de cada nodo del cluster sobre el que se quiere realizar el despliegue.
- *ansible_ssh_pass*: Siguiendo en la misma linea , se deberá especificar la contraseña del user introducido en el anterior campo.

Una vez realizada la configuración de este inventario, este contendría toda la información necesaria, posibilitando así que *Ansible* pueda realizar la conexión con cada uno de los equipos del *cluster*.

Ejecutar el *playbook* que crea mismos usuarios en el *cluster*

Algo a tener en cuenta es que en nuestro *cluster* debe existir un usuario común en cada uno de los nodos sobre los que se realizará el despliegue, es por ello que se nos ofrece la posibilidad de automatizar dicho proceso haciendo uso de los *playbooks*. Para ello se ejecutará el siguiente comando desde el terminal:

ansible-playbook -b playbooks/generic/users.yml

Este *playbook* creará por defecto en todos los nodos un usuario ”nvidia” cuya contraseña será ”deepops”. Estos dos parámetros son fácilmente modificables accediendo al archivo de esta ruta:

config/group_vars/all.yml

Conseguir las claves ssh

Algo que tenemos que tener en cuenta también es que *Ansible* tiene que tener acceso mediante SSH a los usuarios creados en cada equipo del *cluster*. Para conseguir esto, deberemos acceder a cada equipo del *cluster* en el usuario sobre el que vamos a realizar el despliegue. En este caso el usuario es *Nvidia* , y mediante el uso del terminal introducir el siguiente comando, el cual nos generará dos ficheros:

ssh-keygen

Dichos ficheros son las claves pública y privada necesarias para establecer conexión. Deberemos copiarlos para seguidamente pegarlos en un directorio en especifico del ordenador sobre el cual estamos haciendo la configuración, dicho directorio puede ser elegido arbitrariamente por el usuario, aunque de normal se utiliza el directorio llamado .ssh ubicado en /home/*user*.

Editar nuevamente el inventario

Una vez se han creado los mismos users sobre todos los nodos del cluster y se han obtenido las claves ssh de estos, se debe proceder modificando los valores descritos a continuacion:

- *ansible_ssh_user*: Como se explicaba en el apartado en el que se creaban los mismos usuarios, se necesita que en los nodos del *cluster* exista un mismo usuario. Es por esto que el usuario que se debe especificar ahora en este campo es el creado anteriormente, cuyo nombre es *“nvidia”*
- *ansible_ssh_pass*: De la misma forma, este campo debe especificar la contraseña del usuario creado sobre todos los nodos, esta es *“deepops”*

Además se deben especificar dos nuevos campos los cuales no eran necesarios anteriormente pero recibirán importancia en el despliegue final, estos son:

- *ansible_ssh_private_key_file*: Aquí se deberá especificar la ruta donde se encuentran almacenadas las claves ssh de los equipos del *cluster* previamente generadas.
- *ansible_sudo_pass*: En este campo se deberá escribir la contraseña utilizada para acceder al modo *sudo*.

Verificar Acceso a todos los nodos

Para asegurarnos que no ha habido ningún error relacionado con el inventario, deberemos verificar si efectivamente *Ansible* tiene acceso via SSH a todos los equipos. Dicho procedimiento se realizará mediante el uso del terminal, el cual tiene que tener su directorio de trabajo ubicado en el repositorio como se especificaba en el apartado 4.1, una forma de comprobarlo es mediante el comando *pwd*.

El comando que utilizaremos para dicha comprobación es:

ansible all -m raw -a "hostname".

Una vez ejecutado el comando, se mostrará por terminal un resumen que indica el número de pasos exitosos y fallidos, por lo que en este caso se puede observar que la comprobación se ha realizado de forma exitosa. Dicho resumen se muestra en la Figura 4.7.

```
(base) cvb-datathon@cvb-datathon:~/Escritorio/deepops$ ansible all -m raw -a "hostname"

PLAY [Ansible Ad-Hoc] *********************************************************************************************

TASK [raw] ********************************************************************************************************
changed: [master-slurm]
changed: [microdraw]

PLAY RECAP ********************************************************************************************************
master-slurm               : ok=1    changed=1    unreachable=0    failed=0    skipped=0    rescued=0    ignore
microdraw                  : ok=1    changed=1    unreachable=0    failed=0    skipped=0    rescued=0    ignore
```

Figura 4.7: Comprobación para verificar el correcto acceso por SSH de *Datathon* a los nodos del *cluster*

Añadir la opción de desplegar OpenOnDemand

Una vez terminado el paso anterior, contamos con todo lo necesario para realizar el despliegue, aunque en este caso se ha decidido que el despliegue también instale *OpenOnDemand*, descrito en el punto 3.7, el cual por defecto está deshabilitada su instalación. Dicho servicio, resulta muy útil a la hora de contar con una interfaz gráfica para el lanzamiento de tareas mediante *Slurm*.

Es por ello que procederemos a habilitar la opción de despliegue, para ello se modificará un *playbook* ubicado en la siguiente ruta desde la carpeta principal del repositorio clonado:

/config/group_vars/slurm-cluster.yml

Una vez abierto el documento mediante nuestro editor de texto preferido ubicaremos la siguiente linea la cual tendremos que cambiarla a *yes* como se muestra en la Figura 4.8.

```
143 ##############################################################################
144 # Open OnDemand                                                              #
145 ##############################################################################
146 install_open_ondemand: yes
147 # OOD Linux-host adapter requires `slurm_cluster_install_singularity` to be true
148 ood_install_linuxhost_adapter: no
149
150 servername: '{{ ansible_fqdn }}'
151 httpd_port: 9050
152 httpd_listen_addr_port:
153   - 9050
154 httpd_use_rewrites: false
155 node_uri: /node
156 rnode_uri: /rnode
157
```

Figura 4.8: Modificación de la variable para activar el despliegue de *OnDemand*

Lanzar el despliegue

Después de este último cambio ya tendríamos completamente preparado el despliegue de los diferentes *softwares* sobre el *cluster* mediante *Ansible*, por lo que procederemos a ejecutar el siguiente comando, el cual pondrá inicio al despliegue:

ansible-playbook -l slurm-cluster playbooks/slurm-cluster.yml -vvv

Para la ejecución de este *playbook*, el cual dará comienzo al despliegue, se recomienda el uso del modo *verbose* añadiendo -vvv al final del comando, el cual no ofrecerá un mayor detalle de los procesos que se están llevando a cabo, así como una mayor información de posibles errores que puedan surgir durante el despliegue. Dicho proceso tendrá una duración bastante mas notoria en comparación con la comprobación de conexión realizada en 4.1, ya que de forma automática realiza una gran cantidad de comprobaciones, instalaciones y configuraciones de todo tipo.

Una vez terminado dicho proceso, se nos mostrará en la terminal un resumen, como en la Figura 4.9, el cual mostrará cuantos procesos se han llevado a cabo y en que estado han sido finalizados, es por eso que, al igual que en el apartado 4.1, se deberá prestar especial atención a la variable *failed*, dado que el despliegue se realiza de forma secuencial y paralela entre los nodos, esto posibilita que se pueda encontrar en la situación en la que ciertos nodos finalicen el despliegue de forma exitosa y otros no.

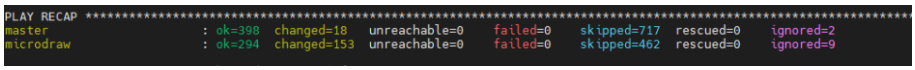

Figura 4.9: Resumen del despliegue

Resumen de procesos realizados

De forma esquemática en la Figura 4.10 se muestran los procesos que se han llevando a cabo para el desarrollo de un sistema para la gestión de colas

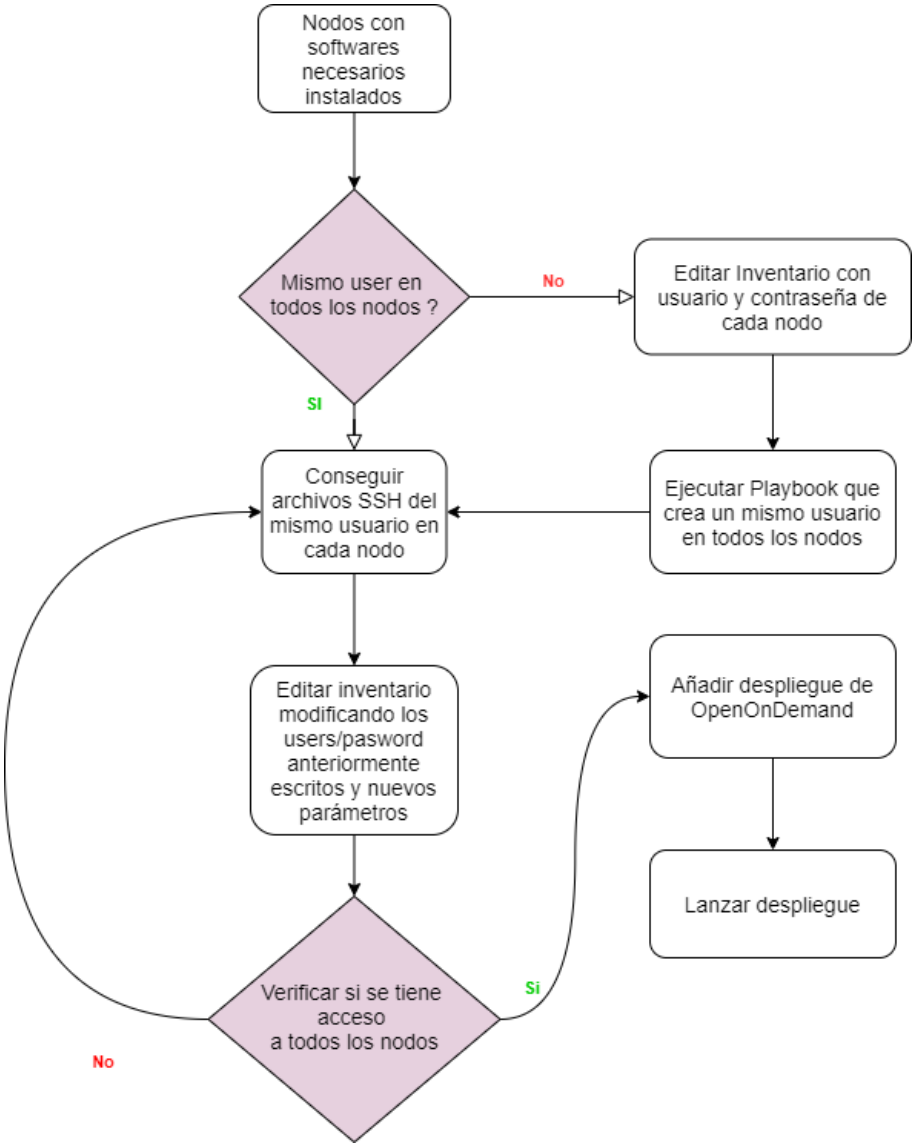

Figura 4.10: Diagrama explicativo de los pasos llevados a cabo

4.2. Comprobar los servicios

A continuación procederemos a verificar el correcto funcionamiento de los *softwares* desplegados por todo el *cluster*, los cuales nos permitirán el envío de tareas mediante *OpenOnDemand* así como una monitorización constante tanto de los nodos como de las tareas enviadas gracias a *Graphana* y *Prometheus*.

4.2.1. Slurm

Empezaremos por *Slurm* (ver apartado 3.1) el cual recordamos que es el software gracias al cual es posible encolar y ejecutar las tareas en los nodos seleccionados aunque sin ofrecernos una

interfaz gráfica , es decir se realiza todo a través del terminal.

Una forma de comprobar que *Slurm* se está ejecutando correctamente es consultando el estado de las particiones de nuestro *cluster*, que por defecto será una única partición de nombre *batch*. Cabe destacar que desde el archivo *slurm.conf* se nos ofrece la posibilidad de modificar las particiones así como añadir tantas como sean necesarias.

En la Figura 4.11 se puede observar como de forma simple se nos muestra información del numero de nodos que componen dicha partición así como su disponibilidad y su estado actual. En el caso de la Figura 4.11, el estado del nodo es *idle*, es decir , se encuentra inactivo ya que no está ejecutando ningún tipo de trabajo y por tanto está disponible.

```
(env) master-ans@master:~/Desktop/deepOps$ sinfo
PARTITION AVAIL  TIMELIMIT  NODES  STATE NODELIST
batch*       up   infinite      1   idle microdraw
```

Figura 4.11: Comprobación del estados de las particiones

Si se desease obtener una mayor información, tanto de la partición como de un nodo en concreto, también se dispone de los siguientes comandos con los que conseguiremos ese grado de detalle deseado (ver Figura 4.12 y Figura 4.13).

scontrol show node microdraw

scontrol show partition batch

```
(env) master-ans@master:~/Desktop/deepOps$ scontrol show node microdraw
NodeName=microdraw Arch=x86_64 CoresPerSocket=3
   CPUAlloc=0 CPUTot=6 CPULoad=0.69
   AvailableFeatures=(null)
   ActiveFeatures=(null)
   Gres=gpu:2
   NodeAddr=microdraw NodeHostName=microdraw Version=20.11.3
   OS=Linux 5.8.0-50-generic #56~20.04.1-Ubuntu SMP Mon Apr 12 21:46:35 UTC 2021
   RealMemory=1000 AllocMem=0 FreeMem=21555 Sockets=1 Boards=1
   State=IDLE ThreadsPerCore=2 TmpDisk=0 Weight=1 Owner=N/A MCS_label=N/A
   Partitions=batch
   BootTime=2021-05-11T16:28:08 SlurmdStartTime=2021-05-11T16:33:46
   CfgTRES=cpu=6,mem=1000M,billing=6,gres/gpu=2
   AllocTRES=
   CapWatts=n/a
   CurrentWatts=0 AveWatts=0
   ExtSensorsJoules=n/s ExtSensorsWatts=0 ExtSensorsTemp=n/s
   Comment=(null)
```

Figura 4.12: Especificaciones del nodo

```
nvidia@microdraw:~$ scontrol show partition batch
PartitionName=batch
   AllowGroups=ALL AllowAccounts=ALL AllowQos=ALL
   AllocNodes=ALL Default=YES QoS=N/A
   DefaultTime=NONE DisableRootJobs=NO ExclusiveUser=NO GraceTime=0 Hidden=NO
   MaxNodes=UNLIMITED MaxTime=UNLIMITED MinNodes=0 LLN=NO MaxCPUsPerNode=UNLIMITED
   Nodes=microdraw
   PriorityJobFactor=1 PriorityTier=1 RootOnly=NO ReqResv=NO OverSubscribe=NO
   OverTimeLimit=NONE PreemptMode=OFF
   State=UP TotalCPUs=6 TotalNodes=1 SelectTypeParameters=NONE
   JobDefaults=(null)
   DefMemPerCPU=UNLIMITED MaxMemPerNode=UNLIMITED
```

Figura 4.13: Especificaciones de la partición

4.2.2. OpenOnDemand

En este caso procedemos a comprobar *OpenOnDemand* el cual añade un portal web a través del cual se pudieran encolar trabajos al *cluster* a través de *Slurm* haciendo uso de una interfaz gráfica.

Es por ello que la forma de comprobar su correcto funcionamiento es accediendo a dicho portal escribiendo la IP del nodo que realiza la función de *Master*, seguida del numero de puerto *9050*, separado mediante dos puntos de la siguiente forma:

http://Ip:9050/

Seguidamente visualizaremos un cuadro emergente el cual nos pedirá las credenciales que necesitaremos introducir cada vez que queramos acceder a la plataforma, estas son:

- Usuario: *nvidia*
- Contraseña: *deepops*

4.2.3. Prometheus

Una vez el despliegue se realice de forma satisfactoria se encontrará, en cada uno de los nodos de cómputo, un *docker* en ejecución perteneciente a este servicio, el cual será accesible por HTTP introduciendo en el buscador deseado la dirección IP del nodo especificado como *"Master"* en el inventario de *Ansible*, previamente descrito en el apartado 3.2.1, seguido del puerto en el que se ha desplegado el sistema, en este caso se ubica en el puerto 9090.

Accediendo a la página de *Prometheus*, se puede encontrar que esta ofrece la posibilidad de realizar consultas sobre el estado de diferentes métricas tales como: temperatura de GPU, potencia utilizada por cada una de las GPU, utilización de la capacidad de GPU expresada en porcentaje, etc. Para consultar dichas variables se debe realizar una consulta con variables tales como: *"DCGM_FI_DEV_GPU_TEMP"* la cual está asociada con la temperatura de las GPU por ejemplo. Dicha variable se consulta en la Figura 4.16 para poder visualizar así la medición de la temperatura en las 2 GPU que dispone el nodo de cómputo.

Figura 4.14: Medición de temperatura en las 2 GPU del nodo

4.2.4. Grafana

A la hora de acceder a dicho servicio, se realizará de forma semejante al apartado 4.2.2 aunque, en este caso se variará el puerto con el que se accede ya que al realizar el despliegue dicho servicio se despliegue en el puerto 3000 por defecto, por lo que el *link* que dará acceso a la plataforma es el siguiente:

http://Ip:9050/

Una vez accedido al servicio, se mostrará una primera página la cual ofrece la elección del *dashboard* que se desea visualizar como se puede observar en la Figura 4.15.

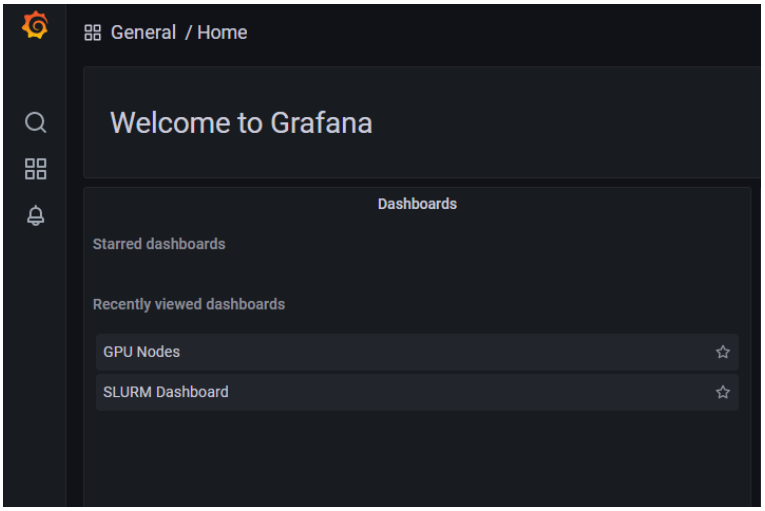

Figura 4.15: Página principal de Grafana

En este caso se contará con dos *dashboards* a visualizar, los cuales se explicarán con mayor detalle a continuación, ambos ofrecen la posibilidad de modificar el rango temporal que se desea visualizar, pudiendo elegir hasta una resolución de 1 año o un rango personalizado entre dos fechas si así se desea, además también cuenta con diferentes intervalos de actualización siendo 30s el establecido por defecto.

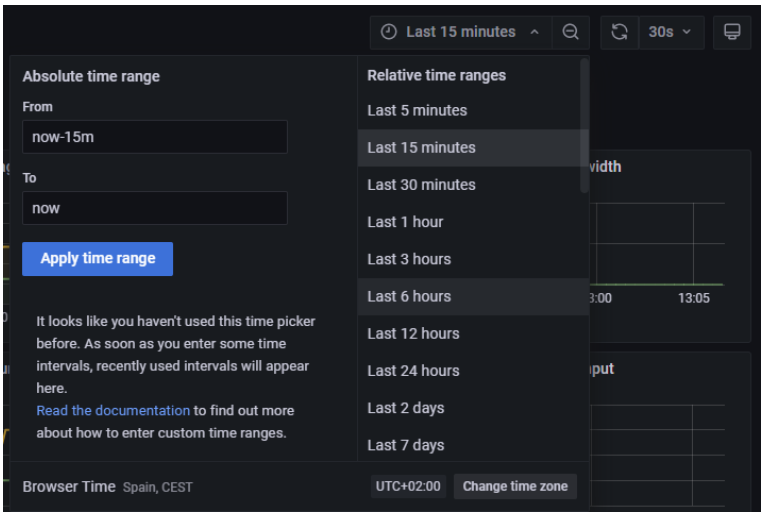

Figura 4.16: Modificación rango temporal

Dashboard GPU

En este tablero se nos mostrará información de interés relacionada con las GPU consiguiendo que se pueda visualizar de forma clara todas aquellas variables tales como la temperatura de estas , el uso de potencia, la memoria usada, etc.

Este tablero suele resultar de ayuda para comprobar cuanto estamos forzando el sistema a la hora de someterlo a fuertes cargas de trabajo y tener un feedback en tiempo real de este.

Figura 4.17: Tablero GPU

Dashboard Slurm

Este tablero se centra más en la monitorización del programa Slurm con el cual conseguiremos tener información en tiempo real del estado de los nodos de computo (slaves) así como de la situación de las tareas enviadas a los nodos pudiendo comprobar si están en espera, ejecución, finalizadas, etc. en varias gráficas temporales.

En la Figura 4.18 se muestran cuatro gráficas, en las que se observa que las dos superiores hacen referencia al estado de los nodos mientras que las otras dos gráficas restantes hacen alusión al estados de las tareas enviadas.

También se puede observar que haciendo esta vez una división vertical, encontramos a nuestra izquierda información relacionada con el correcto funcionamiento tanto de los nodos, como pueden ser los estados: en espera, reservado o completando, como de las tareas encontrando los estados: pendiente, en ejecución o finalizada. Por el contrario al lado derecho de la subdivisión, encontraremos representados estados no deseados tanto de los nodos, como podrían ser estados de *down* o *drained*, como de las tareas enviadas, si estas han tenido algún fallo, han sido canceladas, suspendidas o excedió su tiempo de ejecución.

Sabiendo interpretar las gráficas se puede observar que, en el intervalo de visualización seleccionado, el nodo se ha encontrado en estado *Down* durante un intervalo no superior a media hora y también que una tarea lanzada sobre las 15:30 ha fallado a la hora de ser ejecutada.

Figura 4.18: Tablero Slurm

4.3. Ejecución de una tarea de IA

A continuación ejecutaremos una tarea de IA que mediante el uso de redes neuronales convolucionales, cuya topología se puede observar en la imagen 2.5, sea capaz de aprender a distinguir el contenido de diez tipos de imágenes, las cuales serán facilitadas por un *dataset* que recibe el nombre de *CIFAR-10*.

Este contiene una recopilación de imágenes a color, las cuales son utilizadas, entre otras cosas, para tareas de inteligencia artificial ,más concretamente, algoritmos de visión artificial. Este *dataset* nos ofrece un total de 60.000 imágenes son resolución de 1024 píxeles (32x32), dichas imágenes están agrupadas en diez tipos diferentes dependiendo que se esté fotografiando. Cada uno de estos tipos de imágenes cuentan con 6000 fotografías y representan perros, aviones, camiones, barcos, gatos, coches, caballos, ciervos, pájaros y ranas.

En la Figura 4.19 se puede observar una muestra de 20 de las imágenes que se pueden encontrar en dicha recopilación.

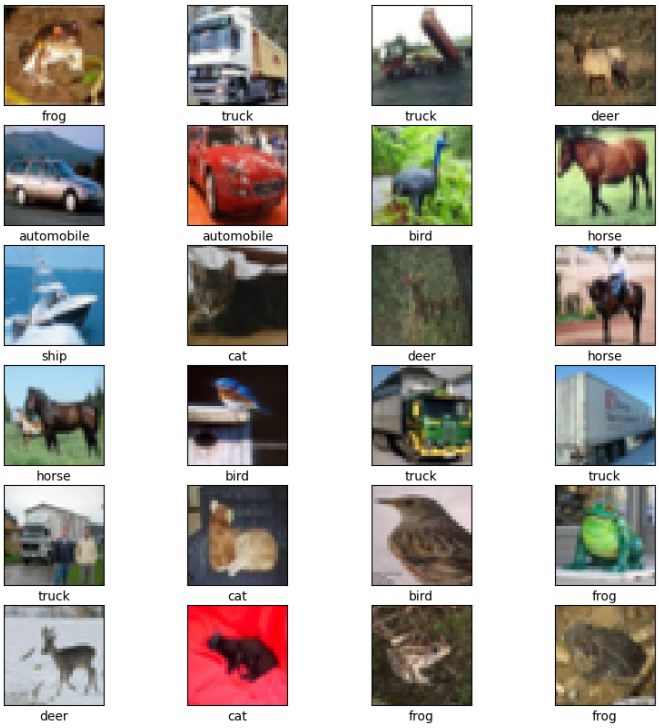

Figura 4.19: Imágenes pertenecientes al dataset CIFAR-10

Para el envío de dicha tarea se utilizará una CNN, la cual consta de dos capas intercaladas múltiples veces, estas capas son *conv2D*, explicada con mayor detalle en la Figura 2.6, y *maxpooling2d* la cual se detalla en la Figura 2.7. Además encontramos otras dos capas más en esta topología las cuales se llaman: *flatten* y *dense*. En la Figura 4.20 se muestra de forma esquemática la topología de esta red neuronal convolucional en particular.

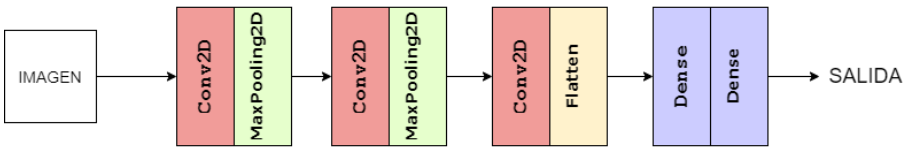

Figura 4.20: Topología esquematizada de la CNN utilizada

A la hora hablar de hiperparámetros, los cuales se definen mediante variables que permiten ajustar diferentes aspectos del periodo de entrenamiento. Se ha decido limitar las épocas a 10 durante, las cuales la red tratará de aprender con 50000 imágenes del *dataset* CIFAR-10. A la hora de hablar de tasa de aprendizaje, se debe resaltar que esta red cuenta con un optimizador "Adam" el cual por defecto cuenta con un valor de 0.001.

Entrando en mas detalle, se va a proceder a describir la funcionalidad de las lineas de código utilizadas en la tarea de inteligencia artificial.

En la Figura 4.21, se muestra la primera parte del código, en esta se importan las librerías de *tensorflow*, de las cuales se obtienen los datasets o los modelos entre otros, así como de matplotlib, utilizado para la representación en gráficas. Adicionalmente se descargará y preparará el dataset CIFAR-10.

```
import tensorflow as tf

from tensorflow.keras import datasets, layers, models
import matplotlib.pyplot as plt



(train_images, train_labels), (test_images, test_labels) = datasets.cifar10.load_data()


train_images, test_images = train_images / 255.0, test_images / 255.0
```

Figura 4.21: Se importan las librerías necesarias y el dataset

A continuación,como se muestra en la Figura 4.22, se realizará una representación de una parte de las imágenes de entrada para comprobar que se visualicen de forma correcta, dicha representación se puede visualizar en la Figura 4.19.

```
class_names = ['airplane', 'automobile', 'bird', 'cat', 'deer',
               'dog', 'frog', 'horse', 'ship', 'truck']


plt.figure(figsize=(10,10))
for i in range(24):
    plt.subplot(6,4,i+1)
    plt.xticks([])
    plt.yticks([])
    plt.grid(False)
    plt.imshow(train_images[i])

    plt.xlabel(class_names[train_labels[i][0]])
plt.show()
plt.savefig('Reconocimiento')
```

Figura 4.22: Representación de parte de las imágenes de entrada

Seguidamente en la Figura 4.23 se muestra el proceso para detallar la topología de la CNN, la cual se ha explicado de forma esquemática en la Figura 4.20.

```
model = models.Sequential()
model.add(layers.Conv2D(32, (3, 3), activation='relu', input_shape=(32, 32, 3)))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), activation='relu'))

model.summary()

model.add(layers.Flatten())
model.add(layers.Dense(64, activation='relu'))
model.add(layers.Dense(10))

model.summary()
```

Figura 4.23: Especificación de la topología de la red

Una vez detallada la tipología, se procederá a ejecutar el entrenamiento especificando el optimizador a usar,el cual como se ha comentado anteriormente se trata de "adam", así como las épocas, limitadas a 10, a las cuales se someterá la red en dicho proceso. Esto se puede observar en la Figura 4.24.

```
model.compile(optimizer='adam',
              loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
              metrics=['accuracy'])

H = model.fit(train_images, train_labels, epochs=10,
                    validation_data=(test_images, test_labels))
```

Figura 4.24: Ejecución del entrenamiento especificando hiperparametros

Por último, se procede en la Figura 4.25 a crear una gráfica, la cual muestre diferentes variables que representan la calidad de dicha tarea de entrenamiento.

```
plt.figure(figsize=(10,10))
# Mostramos gráfica de accuracy y losses
plt.style.use("ggplot")
plt.figure()
plt.plot(np.arange(0, 10), H.history["loss"], label="train_loss")
plt.plot(np.arange(0, 10), H.history["val_loss"], label="val_loss")
plt.plot(np.arange(0, 10), H.history["accuracy"], label="train_acc")
plt.plot(np.arange(0, 10), H.history["val_accuracy"], label="val_acc")
plt.title("Training Loss and Accuracy")
plt.xlabel("Epoch #")
plt.ylabel("Loss/Accuracy")
plt.legend()
plt.savefig('Grafica')

test_loss, test_acc = model.evaluate(test_images,  test_labels, verbose=2)

print(test_acc)
```

Figura 4.25: Programación de una grafica que muestra la calidad del entrenamiento

Capítulo 5

Resultados

En este apartado se comprobará los resultado obtenidos a la hora ejecutar la tarea de IA, que se explica con mayor detalle en el apartado 4.3, así como diversa información obtenida gracias a los servicios desplegados.

Primeramente ingresaremos en la plataforma de *OpenOnDemand* para lanzar la ejecución de la tarea, la cual se muestra en la Figura 5.1 y en la Figura 5.4, que de inicio al *script* en *python*. En esta tarea cabe destacar que se especifica el numero de GPU, que procesarán dicha tarea, mediante *–gres=gpu:2*. También se hace uso del comando *sudo nvidia-docker exec dockerMontado python CNN/script*, el cual es propio de *Docker* y ejecutará en el contenedor "dockerMontado.[el] *script* en *python* ubicado el la ruta especificada.

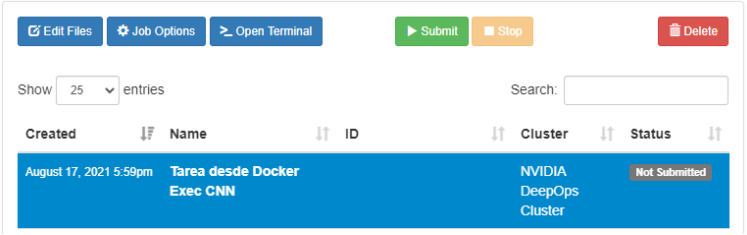

Figura 5.1: Envio de tarea al nodo

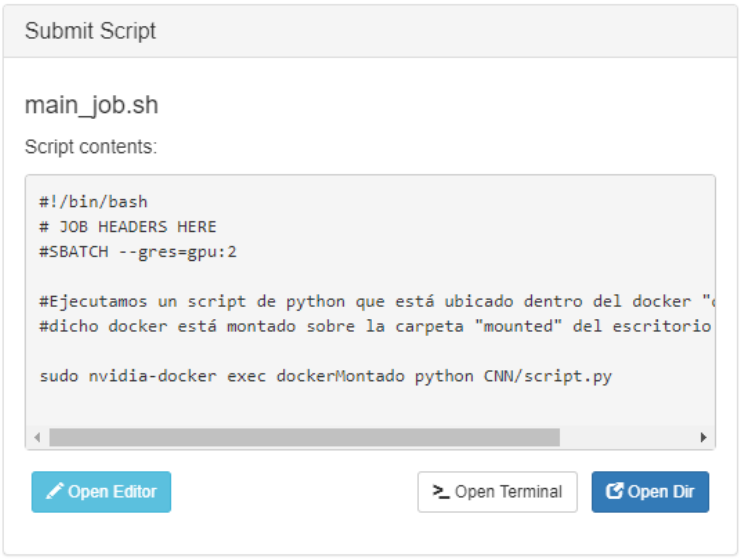

Figura 5.2: Script que ejecutará el código en python

Durante la ejecución de la tarea de inteligencia artificial se puede observar desde *grafana* el estado del nodo, viendo como deja de estar en estado *idle* mientras ejecuta la tarea para volver a ese estado una vez terminada , también se puede observar en la segunda gráfica de la captura el estado de los *jobs* y como primeramente se nos muestra que hay una tarea en ejecución para seguidamente una vez esta ha terminado mostrarnos que ha sido completada.

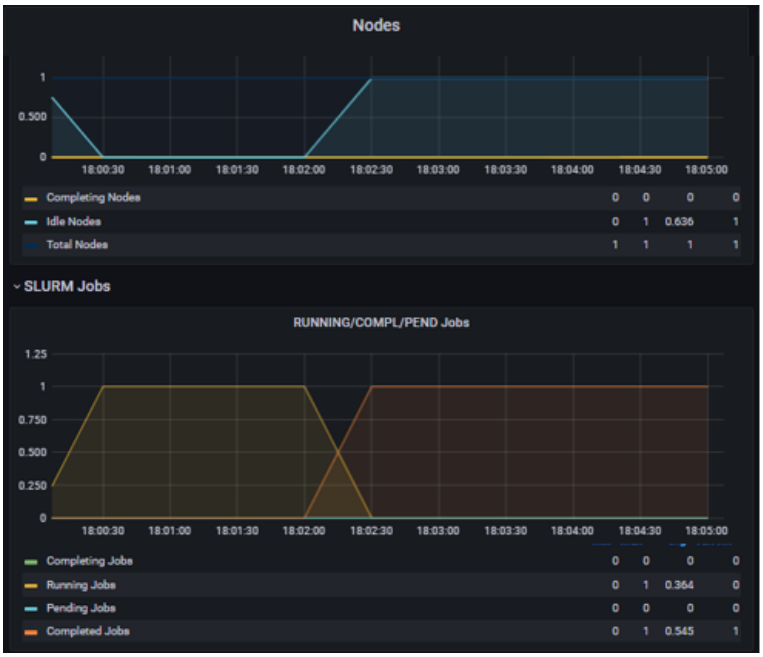

Figura 5.3: Monitorización de la tarea de IA

Además , como se explicó en el punto 4.2.4 se puede visualizar el estado de nuestras GPU

durante la ejecución de la tarea pudiendo ver un incremento en la carga del sistema , el uso de potencia en las GPU e incluso en la temperatura de estas como se muestra a continuación.

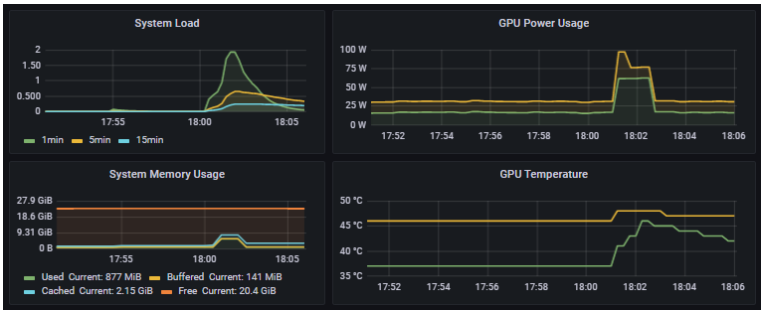

Figura 5.4: Monitorización de las variables del nodo de computación

Una vez finalizada la ejecución podremos ver que se han generado dos imágenes con los resultados del entrenamiento y el resultado del reconocimiento de las imágenes.

En el resultado del entrenamiento se puede observar diversas variables que reflejan el estado de este, entre las cuales se puede destacar *train_acc* el cual hace alusión a la precisión de acierto en el reconocimiento de imágenes, esta aumenta a medida que avanzan las diversas épocas de entrenamiento.

Figura 5.5: Evolución de la IA según las épocas

Como se ha podido comprobar anteriormente el despliegue de los servicios se ha podido realizar correctamente pudiendo utilizarse para el envío de tareas de inteligencia artificial.

Es por esto que, se va a proceder a realizar el despliegue en la que va a ser la verdadera infraes-

tructura de Inteligencia artificial, ya que anteriormente se estaban utilizando los equipos Deployment 3.8.1, Master 3.8.2 y Microdraw 3.8.3 como prueba.

Como podremos observar en la figura 5.6, la nueva infraestructura contará con dos nodos de cómputo llamados *"cvblab03"* y *"cvblab04"*, en lugar de uno. También se mantiene el uso del resto de ordenadores involucrados anteriormente, es decir Master 3.8.2 y Deployment 3.8.1.

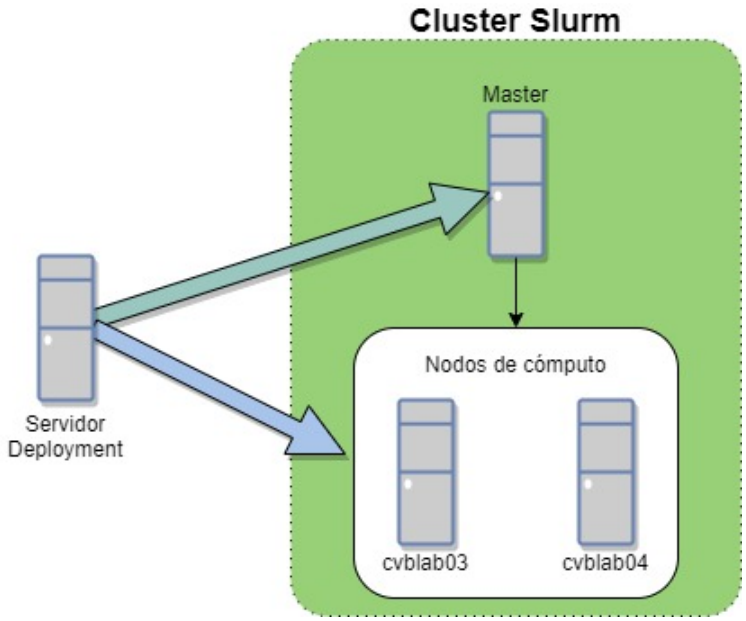

Figura 5.6: Infraestructura final sobre la que se desplegarán los servicios

Una vez realizado el despliegue de nuevo, se puede realizar la consulta *sinfo*, esto detallará el estado de las particiones del cluster y se especificaran los nodos de cómputo que forman parte de dicha partición como se muestra en la Figura 5.7.

```
nvidia@master-slurm:~$ sinfo
PARTITION AVAIL  TIMELIMIT  NODES  STATE NODELIST
batch*       up   infinite      2   idle cvblab[03-04]
```

Figura 5.7: Consulta de estado de las particiones

Adicionalmente, podremos acceder a los servicios anteriormente explicados. En el caso de *Grafana*, podremos elegir mediante un desplegable el nodo que queremos monitorizar sin importar el *dashboard* en el que nos encontremos como se muestra en la figura 5.8.

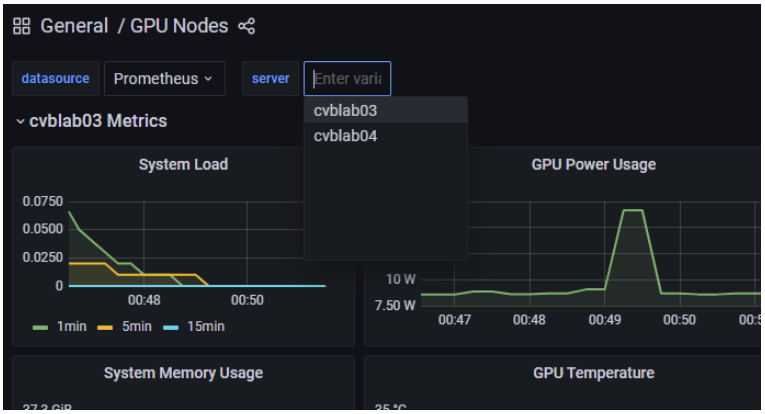

Figura 5.8: Selección de nodo

Capítulo 6

Conclusiones y lineas futuras

Como conclusión se puede destacar que se ha podido llevar a cabo el despliegue del protocolo de gestión inteligente de colas, con el cual se ha permitido realizar el envío de diferentes tareas a los nodos del *cluster*, esto ha permitido el aprender sobre el funcionamiento de estas agrupaciones de equipos conociendo sus diferentes tipos entre otras cosas. También se ha desplegado de forma exitosa dos servicios de monitorización para nuestro *cluster*, los cuales son *Grafana* y *Prometheus*, con estos se ha podido consultar en tiempo real información de cada uno de los nodos de computo del *cluster*, Dicha monitorización ha sido de utilidad tanto para controlar la temperatura de dichos nodos, evitando temperaturas excesivas, así como el estado de las tareas que se han enviado pudiendo consultarse de forma muy rápida y gráfica.

Adicionalmente se ha ejecutado exitosamente una tarea de IA especializada en la clasificación de imágenes de un *dataset* dado. Esto ha permitido conocer aspectos generales como: definición y aplicaciones de la IA, operaciones realizados en dos de sus capas así como que ventajas presenta la GPU frente a la CPU a la hora de ejecutar dichas tareas. Así como conocer aspectos más centrados en la tarea ejecutada como puede ser la topología de la red neuronal a entrenar o la funcionalidad detallada de las lineas de código del *script*.

Respecto a las lineas futuras, según se iba avanzando en la elaboración de este TFG han surgido ideas interesantes con las cuales se podría mejorar la utilización de los servicios realizados en el *cluster*. Estas lineas futuras se ven explicadas a continuación:

Administración de users a OOD

Actualmente el despliegue está realizado con un único nodo *master* el cual solo tiene un usuario en la plataforma *OpenOnDemand* eso significa que todas aquellas personas que se conecten compartirán usuario y por tanto se podrán ver las tareas que encolan todos ellos , es por eso que se plantea la posibilidad si es posible crear varios usuarios dentro de la plataforma de forma que cada uno de ellos no puedas ver las tareas que encolan el resto o al menos no poder ver el código que se quiere ejecutar.

Hacer un despliegue mayor con la DGX-A100

Como despliegue final se pretende realizar el despliegue sobre la DGX-A100 una máquina altamente orientada a IA con una gran capacidad de cómputo la cual es la máquina principal del laboratorio de investigación *CVBLab*.

Configurar la QoS de Slurm

También se podría profundizar más en la configuración de *Slurm* y aunque este cuenta con una configuración por defecto que permite la correcta entrega de tareas a los nodos es cierto que su capacidad de configuración es mayor pudiendo añadir opciones propias de QoS de forma que automáticamente se puedan balancear las cargas evitando así una descompensación de trabajo en los nodos.

Bibliografía

[1] Murray Campbell, A.Joseph Hoane y Feng-hsiung Hsu. “Deep Blue”. En: *Artificial Intelligence* 134.1 (2002), págs. 57-83. ISSN: 0004-3702. DOI: https://doi.org/10.1016/S0004-3702(01)00129-1. URL: https://www.sciencedirect.com/science/article/pii/S0004370201001291.

[2] Bowen Baker y col. “Emergent Tool Use From Multi-Agent Autocurricula”. En: *CoRR* abs/1909.07528 (2019). arXiv: 1909.07528. URL: http://arxiv.org/abs/1909.07528.

[3] Christopher Berner y col. “Dota 2 with Large Scale Deep Reinforcement Learning”. En: *CoRR* abs/1912.06680 (2019). arXiv: 1912.06680. URL: http://arxiv.org/abs/1912.06680.

[4] Ping Xu, Chunquan Xu y Aiqun Zheng. “Virtual Passive Bipedal walking on the Irregular Ground using Kinetic Energy Tracking Control”. En: *2020 IEEE International Conference on Mechatronics and Automation (ICMA)*. 2020, págs. 844-849. DOI: 10.1109/ICMA49215.2020.9233847.

[5] Petronilla Battista y col. “Artificial intelligence and neuropsychological measures: The case of Alzheimer’s disease”. En: *Neuroscience & Biobehavioral Reviews* 114 (2020), págs. 211-228. ISSN: 0149-7634. DOI: https://doi.org/10.1016/j.neubiorev.2020.04.026. URL: https://www.sciencedirect.com/science/article/pii/S0149763420303912.

[6] Carl Vondrick y col. “Tracking Emerges by Colorizing Videos”. En: *CoRR* abs/1806.09594 (2018). arXiv: 1806.09594. URL: http://arxiv.org/abs/1806.09594.

[7] Ashish Vaswani y col. “Attention is All you Need”. En: *Advances in Neural Information Processing Systems*. Ed. por I. Guyon y col. Vol. 30. Curran Associates, Inc., 2017. URL: https://proceedings.neurips.cc/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf.

[8] Rajat Raina, Anand Madhavan y Andrew Y. Ng. “Large-Scale Deep Unsupervised Learning Using Graphics Processors”. En: *Proceedings of the 26th Annual International Conference on Machine Learning*. ICML ’09. Montreal, Quebec, Canada: Association for Computing Machinery, 2009, págs. 873-880. ISBN: 9781605585161. DOI: 10.1145/1553374.1553486. URL: https://doi.org/10.1145/1553374.1553486.

[9] Zhe Fan y col. “GPU Cluster for High Performance Computing”. En: *SC ’04: Proceedings of the 2004 ACM/IEEE Conference on Supercomputing*. 2004, págs. 47-47. DOI: 10.1109/SC.2004.26.

[10] Gordon E. Moore. “Progress in digital integrated electronics [Technical literaiture, Copyright 1975 IEEE. Reprinted, with permission. Technical Digest. International Electron Devices Meeting, IEEE, 1975, pp. 11-13.]” En: *IEEE Solid-State Circuits Society Newsletter* 11.3 (2006), págs. 36-37. DOI: 10.1109/N-SSC.2006.4804410.

[11] *2021 Cloud Report*. 2021. URL: https://www.cockroachlabs.com/blog/2021-cloud-report/.

[12] *SGE vs Slurm comparision*. 2021. URL: https://www.uppmax.uu.se/support/user-guides/sge-vs-slurm-comparison/.

[13] A. M. TURING. “I.—COMPUTING MACHINERY AND INTELLIGENCE”. En: *Mind* LIX.236 (oct. de 1950), págs. 433-460. ISSN: 0026-4423. DOI: 10.1093/mind/LIX.236.433. eprint: https://academic.oup.com/mind/article-pdf/LIX/236/433/30123314/lix-236-433.pdf. URL: https://doi.org/10.1093/mind/LIX.236.433.

[14] Adrián Colomer Granero. *Inteligencia artificial en el sector biomédico*. URL: https://www.quo.es/tecnologia/q2004603196/inteligencia-artificial-sector-biomedico-sistema-sanitario-futuro/.

[15] *Tesla autopilot*. URL: https://www.tesla.com/AI.

[16] *Los 10 modelos más populares de inteligencia artificial*. URL: https://www.clubdetecnologia.net/blog/2018/los-10-modelos-mas-populares-de-inteligencia-artificial/.

[17] *Configuración de Slurm*. URL: https://slurm.schedmd.com/slurm.conf.html.

[18] *Tipos de planificadores*. URL: https://slurm.schedmd.com/sched_config.html.

[19] *How ansible works*. URL: https://www.ansible.com/overview/how-ansible-works.

[20] *Docker: a shipping container for linux code*. URL: https://web.archive.org/web/20130808043357/http://www.linux.com/news/enterprise/cloud-computing/731454-docker-a-shipping-container-for-linux-code/.

[21] *Github*. URL: https://es.wikipedia.org/wiki/GitHub.

[22] *Prometheus*. URL: https://prometheus.io/docs/introduction/overview/.