Capítulo 8

Contexto técnico y tecnológico

El presente constituye un trabajo de fin de grado (TFG) perteneciente a la mención de Computación del Grado en Ingeniería Informática, impartido por la Facultad de Informática de Barcelona.

Durante la planificación de este proyecto, se remarcó la importancia de llevar a cabo un aprendizaje autónomo intensivo respecto a una serie de tecnologías y técnicas con las que se trabajan a lo largo del proyecto. Lo cierto es que varias de estas herramientas no son explicadas a lo largo de la carrera, contexto desde el cual se parte ante la elaboración del trabajo. Por lo tanto, en este capítulo se explicarán a nivel conceptual y técnico algunos de los elementos más importantes que conforman el proyecto, sirviendo como evaluación del aprendizaje autónomo previsto y como guía para posibles lectores del presente documento.

8.1 Machine Learning

El Machine Learning (ML), en español, Aprendizaje Automático, es aquella rama de la Inteligencia Artificial cuyo objetivo es desarrollar técnicas para que las máquinas aprendan por su propia cuenta. Para llevar a cabo este objetivo, se provee de un conjunto de datos a un computador. Generalmente estos datos comparten una serie de atributos, lo que permite al computador identificar patrones y construir y entrenar un modelo con ellos. Dicho modelo le servirá como una hipótesis, para hacer predicciones en el futuro acerca de nuevos datos que compartan los atributos del modelo en cuestión.

Dada la introducción anterior, podríamos llegar a confundir el ML con los modelos estadísticos, ya que comparten ciertas atribuciones. No obstante, cabe destacar que ambos procedimientos tienen diferentes campos de aplicación. En modelación estadística, a partir de una muestra de datos, se formaliza la relación entre sus variables en forma de ecuaciones matemáticas para predecir resultados. A diferencia del ML, se utilizan menos datos y con menos atributos, ya que se requiere que la persona que actue de modelador entienda la relación entre las variables

antes de ingresar el resultado. En cambio, en ML, no se requieren conjeturas anteriores sobre las relaciones entre las variables y se preciden datos sobre la marcha.

Para el tipo de predicciones que queremos llevar a cabo, al igual que pasa con muchas aplicaciones en la actualidad, el Machine Learning nos ofrece una alternativa idónea, ya que nos permite procesar datos con una cantidad mayor de atributos y observaciones. Además el esfuerzo humano queda muchos más reducido, ya que en vez de tener que estudiar relaciones entre variables implicadas, dejamos que el computador haga su trabajo de manera autónoma.

8.2 Federated Learning

8.2.1 GDPR

La nueva legislación GDPR entró en vigor en mayo de 2018 en todos los países de la Unión Europea (UE), y es una actualización importante de la Directiva de protección de datos de la UE introducida en el año 1995. Esta legislación tiene como objetivo proteger los datos personales bajo la perspectiva de que estos solo pueden recopilarse legalmente, bajo condiciones estrictas, para un propósito legítimo. El reglamento completo se describe en detalle en 99 artículos que cubren los principios y los requisitos técnicos y administrativos sobre cómo las organizaciones deben procesar los datos personales.

Para cumplir con los estrictos requisitos del RGPD, las aplicaciones y los servicios convencionales basados en ML deben implementar medidas que protejan y administren de manera efectiva los datos personales, así como también que proporcionen mecanismos para que los interesados controlen por completo sus datos Aunque los sistemas basados en ML se fortalecen con varios métodos de preservación de la privacidad, implementar estas obligaciones en un sistema centralizado basado en ML no es trivial, y a veces tecnológicamente poco práctico.

La recopilación, la agregación y el procesamiento de datos a gran escala en un servidor central en tales sistemas basados en ML no solo implica los riesgos de violaciones graves de datos debido a un único punto de falla, sino que también intensifican la falta de transparencia, el uso

indebido y el abuso de datos. Además, dado que los algoritmos de ML funcionan como una caja negra, también es un desafío proporcionar una interpretación perspicaz de cómo se ejecutan los algoritmos y cómo se toman ciertas decisiones. En consecuencia, en la mayoría de los sistemas basados en ML resulta difícil satisfacer los requisitos de transparencia, equidad y toma de decisiones automatizada del RGPD.

Imaginemos un escenario para ejemplificar un sistema centralizado basado en ML y la importancia del cumplimiento de esta ley: en el campo de la industria sanitaria, el objetivo de poder detectar casos de cáncer de mama se realiza mediante el entrenamiento de modelos de ML, entrenados con datos de pacientes de muchos hospitales. Está claro que existe la necesidad de compartir estos datos, los cuales pueden ser de carácter sensible, para poder entrenar un modelo de ML, agregando datos de los distintos hospitales implicados. Necesitamos aplicar una técnica que nos permita llevar a cabo este objetivo, preservando la privacidad de los datos como estipula la RGDP, y por ende, haciendo un uso responsable y legítimos de los mismos.

8.2.2 Federated Learning

El Federated Learning (FL), en español Aprendizaje Federado, es una técnica de ML que permite entrenar un modelo de ML preservando la privacidad de los datos. Para ello, se requiere de un conjunto de nodos que formen una red, y se hace uso de los diversos nodos para entrenar un modelo de ML común. Cada uno de estos nodos dispondrá de sus datos y entrenará su propio modelo, el cual compartirá al nodo central, que agregará los modelos de los diversos nodos creando así un modelo único, que posteriormente hará llegar a todos los nodos. De esta manera, lo que viaja por la red serán los diversos modelos que se hacen llegar al nodo central o el modelo único agregado, pero nunca los datos concretos que se encuentren en cada nodo.

El Federated Learning nos permite reducir el riesgo de fuga de datos y limita el acceso directo a los mismos. También permite mayores cantidades de datos para entrenar, los modelos pueden generalizar mejor y por lo tanto son más robustos. Finalmente, al llevar el cómputo a diversos nodos se permite realizar un entrenamiento distribuido, otorgando una mayor escalabilidad a la solución. El hecho de llevar a cabo la computación en los nodos en vez de en un servidor

centralizado, es lo que conocemos como Edge Computing, o cómputo en el borde de la red. El aprendizaje federado todavía está en sus fases iniciales y su uso todavía no se ha extendido en la industria, aunque existe una gran cantidad de empresas con interés e impulsando su adopción.

Sin embargo, el aprendizaje federado todavía tiene riesgos abiertos que requieren de un mayor desarrollo y avance en el mismo para convertirse en una solución ideal, por ejemplo:

- Eficiencia de la comunicación: el aprendizaje federado implica numerosas transferencias de datos. En consecuencia, el servidor central o las partes que reciben los parámetros deben ser resistentes a los fallos y retrasos de la comunicación. Garantizar una comunicación y sincronización eficientes entre los dispositivos federados sigue siendo un problema relevante.
- Heterogeneidad del dispositivo: los dispositivos informáticos de las partes federadas son a menudo distintos y a veces desconocidos para las otras partes o el servidor central.
- Heterogeneidad de los datos: los conjuntos de datos de las partes federadas pueden ser muy heterogéneos en términos de cantidad, calidad y diversidad de datos. Es difícil medir de antemano la heterogeneidad estadística de los conjuntos de datos de entrenamiento y mitigar los posibles impactos negativos de esto.
- Privacidad: aunque el propio diseño de FL constituya un avance en el ámbito de la privacidad, existe la necesidad de una implementación eficiente de tecnologías que mejoren la privacidad para evitar fugas de información de los parámetros del modelo compartido.

8.3 Orquestación y contenedores

Este proyecto cuenta con los recursos hardware suficientes como para poder desarrollar y desplegar una aplicación basada en Federated Learning. Se dispone de un testbed desplegado dentro de una red inalámbrica mallada de ciudad, compuesto por diferentes dispositivos de baja capacidad. Como la técnica de FL estipula, en esta aplicación, cada dispositivo cuenta con su base de datos, desde la cual entrena modelos ML. Estos modelos se comparten al dispositivo central, el cual los agrega para crear un modelo central único que es posteriormente compartido a cada dispositivo.

Desde el inicio de este proyecto, se convino que el uso de tecnologías de orquestación de contenedores podría ser una aportación interesante a la aplicación de la que partimos, ya que se intuyó que éstas podrían aportar un mecanismo de control sobre la aplicación de FL. A continuación, se introduce de dónde parten y en qué consisten estas tecnologías y por qué se considera que su uso supone una mejora en el desempeño de la aplicación.

8.3.1 Contenedores

Los contenedores son una pieza de software que contienen todo el código y sus dependencias para que una aplicación puede ser ejecutada, con rapidez y fiabilidad, a partir de la utilización de dicho contenedor.

Esto hace que sean un elemento muy a tener en cuenta de cara a la portabilidad de aplicaciones: si queremos asegurar que una aplicación se ejecute de la misma manera en un dispositvo que en otro, el uso de contenedores nos los garantiza, ya que como hemos dicho, estos contienen todo lo necesario para que una aplicación pueda ejecutarse, independientemente del sistema operativo por dónde corra. Además, otra característica relevante es que son muy ligeros, ya que contienen únicamenente aquello imprescindible para ejecutar la aplicación, lo cual es óptimo a nivel de eficiencia.

Podemos ver, que en una aplicación de Federated Learning, donde dicha aplicación entrena modelos ML en dispositivos potencialmente heterogéneos, el uso de contenedores nos es muy relevante. Su utilización garantiza que en todos los dispositivos se ejecutará la aplicación de la misma manera, y que esta es la forma más eficiente de trasladar la aplicación a los diversos dispositivos.

Aunque no se conciba una alternativa que permita cuestionarnos el uso de contendores para preservar el buen y eficiente uso de la aplicación, si que se considera oportuno su comparación con el uso de máquinas virtuales, ya que este contraste puede ser de ayuda para entender por qué son un mecanismo tan eficiente.

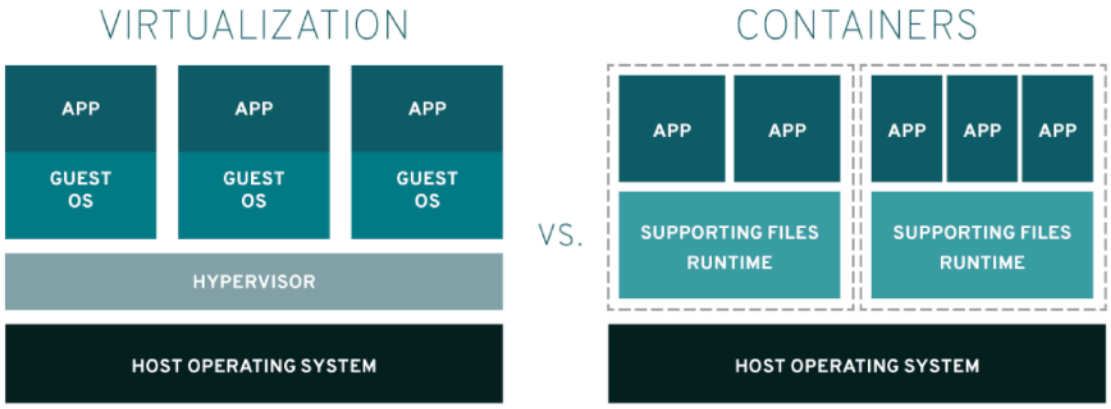

Figura 2: Comparación entre máquinas virtuales y contenedores. Fuente: RedHat

En las máquinas virtuales, las aplicaciones se ejecutan en un sistema operativo instalado sobre otro sistema operativo Host, a partir del uso de un hypervisor. Un ejemplo conocido de hypervisor seria VirtualBox, que nos permite instalar en nuestra máquina otros sistemas operativos y correr aplicaciones dentro de éstos. Los contenedores, en cambio, utilizan muchos menos recursos, ya que aprovecha el kernel del sistema operativo host para ejecutarse. Esto permite además que podamos estar ejecutando múltiples instancias de un contenedor en un dispositivo, ya que éstos usan menos recursos y están completamente aislados entre ellos. Por lo tanto, la mayor eficiencia, portabilidad y escalabilidad de los contenderos nos hacen considerarlos una solución óptima en nuestra aplicación.

8.3.2 Docker

Una vez decidio que sería conveniente el uso de contenedores, nos planteamos utilizar una herramienta que nos permita gestionar contenedores.

El software que nos permite gestionar estos mecanismos es Docker, una tecnología de código abierto gracias a la cual podemos crear y usar contenedores en Linux. Docker utiliza el kernel de Linux y supone una infraestructura basada en imágenes.

Una imagen es un archivo inmutable, una especie de plantilla que se puede utilizar durante la creación de un nuevo contenedor. Un ejemplo básico de imagen seria una distribución concreta de Linux. Además, durante el diseño un contenedor, podemos guardar un estado concreto de éste en una imagen. De esta forma, las imágenes también posibilitan el control de versiones, ya

que podemos guardar múltiples versiones de un contenedor en diferentes imágenes, o en una misma imagen con diversas etiquetas o tags. Las imágenes usan registros, o registries, para almacenarse, de manera que puedan compartirse y reutilizarse.

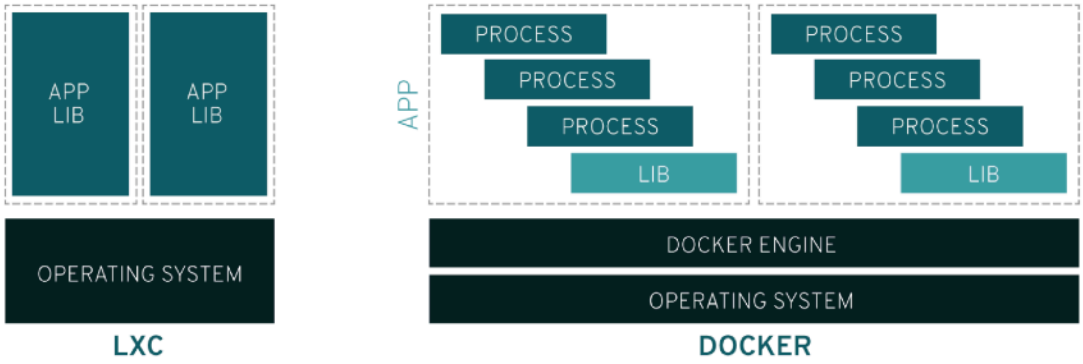

Imagen 3: Arquitecura LXC vs Docker. Fuente: RedHat.

Además, el uso de Docker para el manejo de contenedores, supone una mejora respecto a los contenedores de Linux tradicionales (LXC). El uso de imágenes facilita la creación, control de versiones y compartición de nuevos contenedores. Podemos asemejar el empleo de imágenes en Docker como una nueva capa que se adhiere al contenedor en cuestión. Esto permite la subdivisión de las aplicaciones de los contenedores en subprocesos, lo que facilita la actualización de estas aplicaciones, y posibilita volver a versiones anteriores de la aplicación de una manera muy eficiente. Esta manera de separar las aplicaciones en capas respalda el desarrollo ágil.

Es por lo anterior, que para la aplicación de Federated Learning de este proyecto, se consideró inicialmente el uso de Docker como herramienta de gestión de los diversos contenedores implicados. No obstante, Docker tiene una limitación a tener en cuenta: cuando el número de contenedores y aplicaciones que consideramos va creciendo, se dificulta la gestión y organización. Por este motivo, y en beneficio también de la escalabilidad de la aplicación, se considera oportuno el empleo de una herramienta de orquestación de contenedores como es Kubernetes.

8.3.3 Kubernetes

Kubernetes es una plataforma de código abierto destinada a la gestión y automatización de procesos relacionados con contenedores. Kubernetes se organiza entorno a un clúster, que es

un conjunto de hosts locales o en la nube que, entre otras cosas, ejecutan contenedores de Linux. En nuestra aplicación, el clúster esta formado por el conjunto de dispositivos del testbed, siendo cada uno de ellos un nodo del clúster. Pero para entender mejor esta tecnología, podemos dar un vistazo a su arquitectura:

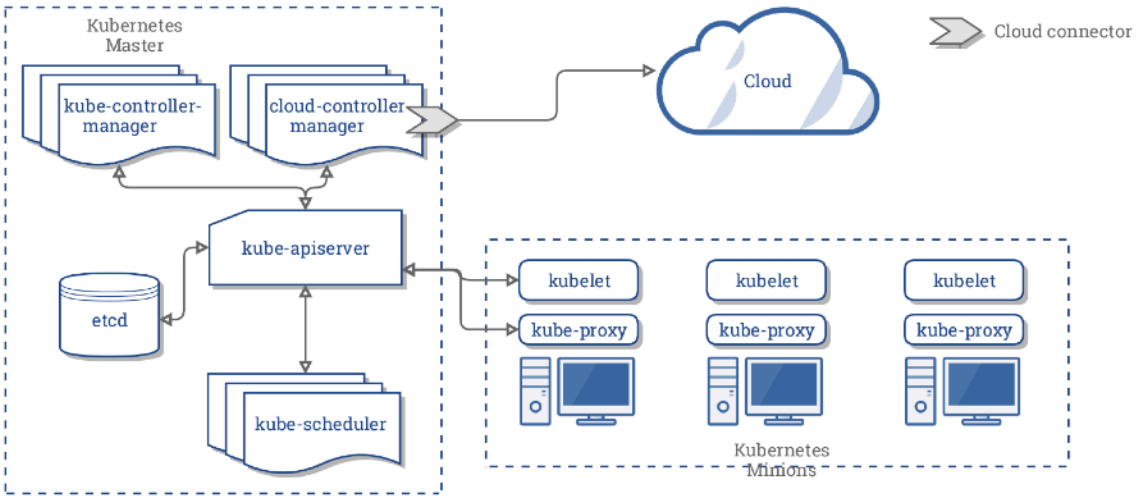

Imagen 4: Arquitectura de Kubernetes. Fuente: Documentación de Kubernetes

Podemos observar una separación entre el Kubernets Master, o plano de control, y los Kubernetes Minions o Workers. El primero es la máquina desde el cual se gestionan al restos de máquinas, virtuales o físicas, denominadas Workers. El Master contiene una serie de servicios adicionales para la orquestación del clúster. Cada uno de los nodos Worker contiene los servicios para ejecutar contenedores.

Dentro del Master encontramos el apiserver, componente que expone la API de Kubernetes y recibe peticiones, y que utiliza la base de datos clave-valor que se halla en etcd para almacenar el estado actual del clúster de manera persistente. El scheduler, por su parte, es el encargado de distribuir la ejecución de los contenedores entre los nodos Worker disponibles, teniendo en cuenta factores de configuración, recursos o restricciones. El objetivo del controller-manager es ejecutar los controladores de Kubernetes necesarios para que el clúster alcance el estado deseado. Finalmente, el cloud-controller-manager se encarga de ejecutar los controladores relacionados con proveedores en el cloud.

Por otros lado, los nodos Worker cuentan con el kubelet, el agente que garantiza la ejecución adecuada de los contendedores en el nodo y el kube-proxy, que garantiza el cumplimiento de

las reglas de red ordenadas por los servicios de Kubernetes. Más adelante en este documento explicaremos en que consisten estos servicios.

Podemos ver que la arquitectura de Kubernetes gestiona el funcionamiento de una infraestructura declarativa, pensada para cumplir un estado deseado del clúster, en el cual se tiene en cuenta la ejecución de contenedores y todos los requisitos implicados. Es por esto que los objetos en Kubernetes varían un poco de los de Docker. En Kubernetes no se ejecutan directamente contenedores, sino pods. Un pod en Kubernetes es un conjunto de contenedores. Generalmente los pods incluyen un único contenedor, como pasará en el caso de nuestra aplicación, pero pueden incluirse más si fuera oportuno. A diferencia de los contenedores en Docker, estos pods son descartables y reemplazables, lo que quiere decir que si por algún motivo, un pod cae o deja de funcionar, Kubernetes iniciaría una nueva instancia de este pod de manera automática. Además, un pod pueda contener varias réplicas de un mismo contenedor, lo que favorece a la escalabilidad y la disponibilidad de las aplicaciones. Al igual que los contenedores en Docker, los pods utilizan imágenes almacenadas en registros como plantilla que les facilita su diseño y modificación. Por lo tanto, similarmente a Docker, vemos que en Kubernetes el código es reutilizable y constituye partes modulares, lo que abre la posibilidad a una arquitectura de microservicios. Esta arquitectura se basa en desarrollar sistemas que sean flexibles, escalables y con un mantenimiento sencillo, y cuya modularidad permita ofrecer aplicaciones formadas por componentes independientes que ejecuten cada proceso de la aplicación como un servicio.

8.3.3.1 Distribuciones

Dadas las características expuestas anteriormente, Kubernetes se considera la plataforma perfecta para alojar aplicaciones nativas de la nube que requieren de alta disponibilidad y de un escalado rápido, ya que los clústeres de Kubernetes pueden abarcar hosts en nubes públicas, locales, híbridas o privadas. Esto ha impulsado a la industria a suministrar diversas distribuciones de Kubernetes, algunas de ellas orientadas a la nube, como las que proveen Amazon, Google o Red Hat. Para nuestra aplicación, requerimos de una distribución liviana que nos permita desplegar un clúster de Kubernetes de una manera relativamente sencilla, ya que estaremos utilizando dispositivos de baja capacidad. Para este propósito, consideramos minikube, microk8s y k3s.

En primer lugar, minikube es una distribución ligera de Kubernetes desarrollada por el proyecto principal de Kubernetes. Es extremadamente liviana y muy fácil de instalar y usar. No obstante, se instala en una máquina local para simular un clúster por medio de máquinas virtuales, pero si queremos utilizar diversos dispositivos no es una opción adecuada. Por su parte, microk8s es una ligera distribución de código abierto de Kubernetes desarrollada por Cannonical. Finalmente, k3s es también una distribución ligera de Kubernetes de código abierto, desarrollada por Rancher. K3s es ideal para dispositivos pequeños como IoT y Edge Computing, ya que el tamaño del binario de instalación es de 40 MB y funciona incluso en dispositivos con pocos recursos como Raspberry Pi's.

Tanto microk8s como k3s son excelentes herramientas para ejecutar versiones minimizadas de Kubernetes en entornos de desarrollo locales, dispositivos en la nube, perimetrales y de IoT. No obstante, k3s ha sido la candidata elegida ya que esta distribucón se centra más en la reducción de recursos y la computación en dispositivos Edge con más restricciones. Dado que en sí contiene todas las funcionalidades de un clúster de Kubernetes de las que queremos hacer uso, k3s se convierte en la distribución de Kubernetes más adecuada para la orquestación de los componentes de nuestra aplicación basada en Federated Learning.