Desarrollo de Servicios Confiables en Kubernetes Utilizando TEEs

Diego Ferreiro Ferrón
Gradiant
dferreiro@gradiant.org

Mario López Feijoo
Gradiant
mlopez@gradiant.org

Iago López Román
Gradiant
ilopez@gradiant.org

Adrián Vázquez Saavedra
Gradiant
avsaavedra@gradiant.org

***Resumen*—La alta demanda de servicios en la nube y la cantidad de datos sensibles que procesan, hace que sea primordial que ofrezcan garantías de confidencialidad e integridad. La computación confidencial propone el uso de Entornos de Ejecución Confiables (TEEs) basados en hardware para limitar el acceso a los datos no cifrados. Estas soluciones resultan ideales para entornos en la nube, en los cuales la infraestructura es confiada a terceros que no siempre son considerados fiables. En el presente trabajo, se ha desarrollado un servicio confiable, basado en el TEE ofrecido por la tecnología SGX, para el mantenimiento predictivo de vehículos. Este servicio ha sido desplegado utilizado la metodología más común seguida por las organizaciones: contenedores y Kubernetes. Los resultados de nuestro trabajo prueban que la computación confidencial es efectiva y necesaria para reforzar la seguridad de los servicios en la nube actuales.**

***Index Terms*—Computación confidencial, Entornos de computación confiable, Kubernetes**

Tipo de contribución: *Investigación en desarrollo (límite 8 páginas)*

I. Introducción

La seguridad de la información ha sido un desafío constante a lo largo de la historia. En respuesta a esta problemática surgió la criptografía, práctica que consiste en ocultar o cifrar información de manera que sólo el destinatario previsto pueda leerla. Esta información puede estar en tres estados, (1) en tránsito, mientras se traslada de un lugar a otro, (2) en reposo, mientras está almacenada, y (3) en uso, durante su procesamiento.

La criptografía clásica típicamente se ha enfocado en los dos primeros estados. No obstante, en los últimos años, han surgido distintos enfoques criptográficos, como el cifrado homomórfico o la computación segura entre múltiples partes, que abordan la problemática del tercer estado. A pesar de ello, estas técnicas criptográficas están lejos de poder ser utilizadas en la práctica en cualquier tipo de escenario debido a sus limitaciones, como la complejidad computacional [1]. Por ello, en la década pasada, se introdujo el concepto de **computación confidencial**, una solución que opta por introducir entornos de ejecución confiables (**TEEs**) basados en hardware. Los TEEs son dispositivos de propósito general que se caracterizan principalmente por proporcionar un área segura aislada en el procesador principal [2]. Estos dispositivos están diseñados para ejecutar aplicaciones en un entorno protegido y sin interferencias, garantizando el aislamiento a través de soporte hardware. En cuanto al rendimiento, si bien depende del TEE utilizado, este suele ser significante superior a los enfoques puramente criptográficos [3].

En el panorama tecnológico actual, las compañías tienden a ofrecer sus servicios en la nube, dadas las ventajas que esta provee. Los entornos *cloud* se distinguen por la dependencia de las empresas hacia los proveedores de infraestructura, una relación que va más allá de la provisión de recursos y se extiende a aspectos críticos como la seguridad. La capacidad de los proveedores para acceder al hardware les confiere la posibilidad de interactuar con cualquier información alojada en sus sistemas. La computación confidencial emerge como una solución ideal en este tipo de entornos, ya que permite que las organizaciones aíslen sus aplicaciones protegiendo tanto sus datos como los de sus clientes, evitando esta dependencia del proveedor de la infraestructura [4].

En el ámbito de la implementación, los contenedores y Kubernetes representan la metodología más común para el despliegue de aplicaciones en la nube [5]. Los contenedores encapsulan las aplicaciones y sus dependencias, permitiendo una ejecución eficiente y aislada, mientras que Kubernetes orquesta estos contenedores, gestionando su despliegue, escalado y operación de manera automatizada.

La combinación de computación confidencial con la contenerización y la orquestación de Kubernetes permitirá ofrecer soluciones robustas en términos de seguridad en la nube. Los contenedores pueden integrarse con los TEEs, proporcionando un entorno de ejecución seguro, y Kubernetes puede asegurar que sólo los contenedores autorizados y verificados se ejecuten, manteniendo la integridad y confidencialidad de los datos en todo momento. Esta sinergia entre la computación confidencial, contenedores y Kubernetes es fundamental para avanzar hacia un modelo de seguridad *zero trust* en la nube [6].

I-A. Nuestra contribución

En este trabajo, hemos diseñado y desarrollado un servicio confiable que se despliega dentro de un TEE para garantizar la confidencialidad e integridad de la información que procesa. El servicio ha sido desarrollado utilizando la tecnología Intel SGX [7], un TEE que permite aislamiento a nivel de aplicación. Es decir, cada aplicación estaría aislada, no sólo de otras aplicaciones que se están ejecutando en el sistema, si no que ni el sistema operativo ni el hipervisor tendrían acceso a la información manejada.

Nuestro servicio ofrece la capacidad de registrar de forma confidencial un modelo de *Machine Learning* (ML) que haya sido entrenado previamente. Posteriormente, permite a los usuarios enviar sus datos, de manera también confidencial, para realizar inferencias empleando este modelo. El servicio también ofrece la funcionalidad de acreditación para que cualquier usuario que se comunique con él pueda verificar que se está ejecutando en el entorno confiable.

ISBN:978-84-09-62140-8

Además, hemos optado por un modelo de arquitectura que se ha convertido casi en un estándar del sector, la utilización de contenedores en Kubernetes. Al contenerizar el servicio, logramos independizar el despliegue de la infraestructura. Por último, el servicio está corriendo en un clúster de Kubernetes con soporte Intel SGX, lo que nos facilita un escalado eficaz y una gestión simplificada.

II. Preliminares

La computación confidencial es una tecnología que garantiza la seguridad de los datos durante el procesado. Para ello, utiliza entornos de ejecución confiables, entornos aislados para computación protegida.

Aunque el término computación confidencial es relativamente nuevo, los dispositivos en los que se basa existen desde hace algunas décadas. Ciertas clases de TEEs han estado disponibles en dispositivos comerciales en formatos como TPMs (Módulos de Plataforma de Confianza) y HSMs (Módulos de Seguridad de Hardware) [8]. Estos dispositivos presentan un marcado enfoque criptográfico, en otras palabras, su función principal es ejecutar operaciones criptográficas evitando que las claves se expongan en entornos no seguros.

No obstante, en los últimos años, se ha desarrollado una nueva categoría de TEEs. Esta categoría de entornos de computación confiables son dispositivos de propósito general que se caracterizan por proporcionar un área segura aislada en el procesador principal [2]. Estos TEEs se distinguen por las siguientes propiedades, aunque no todas las tecnologías TEEs que se mencionarán a continuación las cumplen:

- **Confidencialidad**: se garantiza que únicamente las entidades dentro del entorno seguro tienen acceso a su contenido, sea código o datos.
- **Integridad**: se garantiza que el contenido del TEE no se puede modificar por entidades que se encuentren fuera del entorno seguro.
- **Acreditación** (*attestation*): permite a entidades externas verificar que están interactuando con un entorno confiable.

Es preciso aclarar que los TEEs poseen un mecanismo de intercambio de datos con el exterior en donde la comunicación entre la parte segura y la parte no segura se hace a través de un *buffer* de datos (memoria RAM), el cual es compartido entre ambos entornos. Por ello, la información que se introduce en este buffer debe encontrarse cifrada con una clave perteneciente al entorno seguro.

En el panorama actual, todos los grandes fabricantes de procesadores han integrado los entornos de ejecución segura dentro de sus procesadores. La compañía pionera en este campo es ARM ya que desde la década de los 2000s cuentan con la tecnología ARM Trustzone [9]. Además, recientemente ha desarrollado una versión mejorada llamada ARM CCA [10].

Tras la iniciativa de ARM, otros fabricantes como Intel y AMD han contribuido al desarrollo de la tecnología. En particular, Intel con la introducción de Intel SGX [7], mejorado con SGXv2; y más recientemente, con la presentación de Intel TDX [11]. Mientras que AMD por su lado, con la tecnología SEV [12] y sus actualizaciones, SEV-ES [13] y SEV-SNP [14].

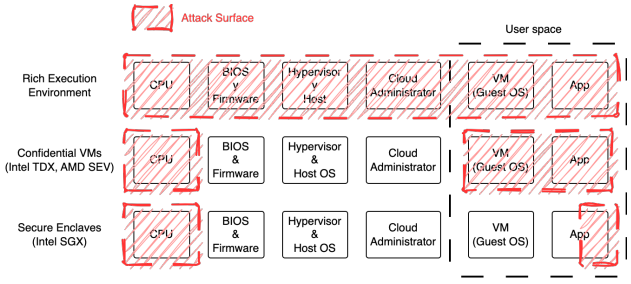

Figura 1. Comparativa superficie de ataque TEEs [4].

Por lo general, podemos dividir todos estos TEEs introducidos en dos variantes, dependiendo del tamaño de la base de cómputo confiable (**TCB**). La TCB es la parte del sistema que implementa la seguridad y confianza. Abarca hardware, software y firmware, siendo esencial para garantizar el funcionamiento seguro del sistema [15]. En otras palabras, en el contexto de los TEEs, es la parte del software que se ejecuta en el entorno seguro, además del firmware/microcódigo del procesador. Considerando este aspecto, tenemos dos variantes: (1) los *secure enclaves*, donde dentro del entorno aislado, únicamente se ejecuta una parte de una aplicación; y (2) las *confidential VMs* o *trust domains*, que ejecutan una máquina virtual completa dentro del entorno seguro.

En la figura 1, se presenta una comparativa entre estas dos variantes en relación al entorno de computación estándar o convencional. Se enfatiza qué entidades tienen acceso a la información en cada caso. Este aspecto está directamente relacionado con los riesgos de seguridad del contenido del entorno seguro. Cuantas menos entidades tengan acceso, menor será el riesgo. Debe destacarse que en este diagrama hemos excluido a aquellas entidades que tienen acceso directo al hardware, esto es debido a que los TEEs no están especialmente diseñados para hacer frente a ataques de canal lateral [16].

Por otro lado, en la tabla I se muestran algunas de las tecnologías anteriormente mencionadas, especificando la variante a la que pertenecen así como las características que ofrecen.

Tabla I
Comparativa propiedades de seguridad TEEs [17].

TEE	Tipo	Confidencialidad	Integridad	Acreditación (Remota)
SGX	*Enclave*	✓	✓	✓
SEV	*VM*	✓*	✗	✗
SEV-ES	*VM*	✓	✗	✗
SEV-SNP	*VM*	✓	✓	✓
TDX	*VM*	✓	✓	✓

Para concluir, aunque inicialmente los entornos de computación confiables se concibieron como áreas seguras dentro del procesador principal, recientemente se ha venido trabajando en extender estas áreas seguras a otro tipo de dispositivos confiables, como el caso de las tarjetas gráficas. Estos dispositivos ofrecen nuevas posibilidades a los entornos de computación confiable, especialmente en aquellos escenarios donde se requiera una gran capacidad de cómputo. En este contexto, NVIDIA está desempeñando un papel clave liderando el desarrollado de esta tecnología con su producto NVIDIA

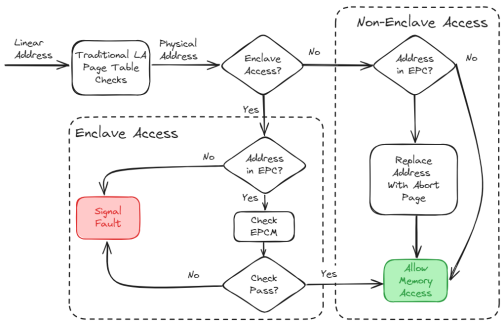

Figura 2. Acceso a memoria con Intel SGX.

Confidential Computing [18].

II-A. Intel SGX

Intel Software Guard Extensions, más conocido como **Intel SGX**, es una tecnología de seguridad integrada en algunos procesadores Intel que garantiza la protección de los datos en uso a través de un aislamiento individual por aplicación. Para ello, permite que las aplicaciones utilicen espacios de memoria cifrados controlados por el procesador conocidos como **enclaves**. El acceso a estos espacios de memoria estaría limitado únicamente a las aplicaciones que los hayan solicitado, ya que el descifrado se hace *al vuelo* en el procesador. La figura 2 presenta un esquema de alto nivel que ilustra cómo el procesador toma decisiones sobre el acceso a una página de memoria.

Intel SGX se integra a nivel del procesador, por lo que las aplicaciones estarán protegidas frente a *malware* y usuarios no autorizados, incluso en caso de vulneración de las capas del sistema operativo o del hipervisor. Se basa en un nuevo conjunto de instrucciones añadidas a la arquitectura de procesadores de Intel en dos fases. El primer subconjunto se añadió en 2015 y es conocido como **SGXv1**. En 2019, para tratar con algunos limitantes de SGXv1, como el tamaño de los enclaves, las operaciones de I/O y ciertos ataques de canal lateral, se introdujo una nueva versión de la tecnología conocida como **SGXv2**.

Todas las aplicaciones que utilizan Intel SGX se dividen en dos partes: una parte confiable y una parte no confiable. La parte confiable comprende todo el código que se carga y ejecuta dentro de la memoria protegida. Por otro lado, la parte no confiable debe, como mínimo, cargar el código que se ejecutará en la parte confiable. Además, tiene la responsabilidad de canalizar toda la información hacia la entidad confiable, dado que la parte confiable carece de acceso a los componentes externos al procesador, como por ejemplo, los dispositivos de red.

Las aplicaciones desarrolladas en base a la tecnología Intel SGX incluyen algunas medidas de seguridad para prevenir ataques en la cadena de suministro. Por esta razón, es imperativo que cualquier aplicación o código que se introduzca en la parte confiable esté debidamente autenticado mediante una firma digital del desarrollador de la aplicación. Sin embargo, investigadores han demostrado que estas protecciones no son completamente efectivas, ya que es posible comprometer el enclave antes de que se firme [19].

El proceso de carga o inicialización del enclave es crucial, ya que garantiza tanto la **integridad** como la **confidencialidad** del código del enclave. Durante el proceso de carga del código en la memoria, se calcula un *hash* criptográfico basado en el contenido del mismo. Este *hash*, posteriormente, se emplea en conjunto con la clave pública del desarrollador para validar la autenticidad de la firma asociada al código. Con respecto a la confidencialidad, de forma transparente al usuario, cada página de memoria del enclave se cifra utilizando el algoritmo AES en modo XTS con el motor criptográfico incorporado en los procesadores Intel una vez que 'sale del procesador'.

Referente a la **acreditación** en Intel SGX, tenemos dos variantes. (1) La **acreditación local** permite que dos enclaves en un mismo servidor puedan establecer una relación de confianza. Este tipo de acreditación utiliza una clave simétrica grabada durante la fabricación del procesador. Los propios enclaves no tienen acceso a la clave pero si pueden solicitar al procesador que les genere una prueba criptográfica llamada *REPORT* que, entre otras cosas, contiene un *hash* del contenido, una referencia a la clave pública del desarrollador y del identificador del micro-código que está utilizando el procesador. El *REPORT* incluye un **MAC** generado con esta clave. De la misma manera que pueden solicitar un *REPORT*, también le pueden pedir al procesador que lo verifique.

(2) La **acreditación remota**, por otro lado, permite que una entidad remota pueda establecer una relación de confianza con un enclave. Si bien existen dos modos de acreditación remota, DCAP y EPID, este último está obsoleto y se dejará de dar soporte por parte de Intel en 2025, por ello describiremos únicamente la acreditación remota DCAP. En la acreditación remota DCAP, al igual que en la acreditación local, se parte de la clave simétrica estampada durante el proceso de fabricación. Sin embargo, en este caso, en lugar de utilizarse directamente, dicha clave simétrica se emplea para autenticarse ante un servicio proporcionado por el fabricante. Este servicio emitirá un certificado para el servidor. De esta forma cuando el enclave necesita acreditarse ante una entidad externa, solicita un *REPORT* al procesador. Este reporte se firmará con el certificado de la plataforma, lo que se conoce como *QUOTE*. La entidad externa tiene la opción de comunicarse con Intel para verificar la *QUOTE*. Finalmente, basándose en el resultado de la verificación de la *QUOTE* y las propiedades incluidas en ella, la entidad externa puede decidir si establecer o no la confianza en el enclave.

En cuanto al soporte de esta tecnología, actualmente está respaldada por los principales sistemas operativos. A partir del kernel 5.11, Linux incluye los *drivers* de SGX en su núcleo. Por su parte, Windows también ofrece soporte para SGX desde la versión 1511 de Windows 10. En lo que respecta a la virtualización, SGX está disponible en la mayoría de las soluciones como KVM/Qemu, VMWare vSphere y Hyper-V, aunque con algunas limitaciones. Por último, en entornos *cloud*, SGX se encuentra disponible en las plataformas de Microsoft Azure, IBM Cloud y Alibaba Cloud.

III. Escenario

El escenario propuesto, consistirá en la creación de un modelo de *Machine Learning* para el mantenimiento predictivo de vehículos [20], y su posterior integración en la plataforma Intel SGX. En primer lugar, para la creación del modelo se

parte de la recopilación de datos de cuatro rodamientos, con una estructura de datos que usa tres conjuntos con valores en los que se describen pruebas de fallo. En segundo lugar, se buscará registrar el modelo y poder realizarle inferencias seguras, tal y como se observa en los pasos uno y ocho de la **figura 3** respectivamente.

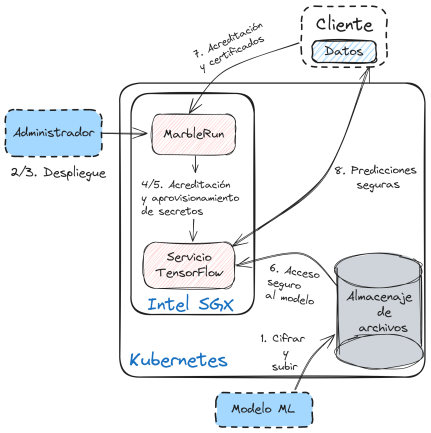

Figura 3. *Privacy Preserving Machine Learning Tensorflow using a* TEE.

A continuación, se procede a dar una descripción superficial de la arquitectura funcional presentada en la **figura 3**. Primero el cliente cifra y sube el modelo ML a una base de datos, posteriormente, se lleva a cabo la autenticación y arranque para poder lanzar el microservicio con su configuración asociada. Acto seguido, el servicio descifra el modelo ML, para luego ejecutarlo y permitir a los clientes realizar inferencias confiables.

IV. Implementación y Despliegue

Para llevar a cabo este apartado, es fundamental contar con un entorno adecuado. Por un lado, necesitamos disponer del hardware correspondiente a la tecnología utilizada, en este caso, Intel SGX. Por otro lado, en lo que respecta al software, es importante destacar que sin las herramientas, *frameworks* y aplicaciones adecuadas, la computación confidencial resulta prácticamente inútil [4]. Estos *SDKs* y *frameworks* permiten construir o adaptar aplicaciones confidenciales, y proporcionan lo siguiente: (1) Abstracción de funciones complejas como la acreditación remota; (2) Empaquetado y firma de aplicaciones (crucial en acreditación remota), y (3) Un entorno de ejecución compatible con la interfaz POSIX (soporte de E/S de red y archivos, multihilo, etc.). Para poder lograr esto, este apartado dará lugar a tres subsecciones. En la primera se hablará de la integración de SGX con Kubernetes. En la segunda subsección se versará sobre el desarrollo del servicio confiable. En la tercera se explicará de manera minuciosa el despliegue del servicio en la plataforma confidencial.

IV-A. Kubernetes con soporte de Intel SGX

Anteriormente se ha hablado de la tecnología Intel SGX, en el punto actual, se va a discutir su integración en entornos de Kubernetes. Para poder llevar a cabo esta integración, en la primera parte de este subapartado, se estudiarán los componentes que lo forman, y en la segunda parte, se dará una explicación de los pasos a tener en cuenta para garantizar su funcionamiento.

Lo primero a considerar en este proceso es que el soporte de la tecnología Intel SGX en un clúster de Kubernetes requiere de tres componentes principales, los cuales están definidos a continuación:

- ***SGX Device Plugin*** [21], cuya responsabilidad radica en descubrir y notificar que nodos tienen un dispositivo Intel SGX. Se debe aclarar que si los contenedores solicitan recursos sobre la tecnología SGX en el clúster, estos no deben utilizar directamente los recursos proporcionados por el *device plugin*.
- ***SGX Admission Webhook***, su función consiste en realizar los cambios necesarios durante la configuración del *Pod*. Para ello emplea el proveedor de *Quotes* de Intel SGX, que tendrá la responsabilidad de lograr el funcionamiento de la acreditación remota en el clúster. Cabe resaltar que este debe establecerlo el usuario mediante una anotación que recibe el nombre de *sgx.intel.com/quote-provider*. Su objetivo principal consiste en ocultar los detalles de configuración de los recursos del dispositivo, así como los de los puntos de montaje de volúmenes que se usen para la acreditación remota de Intel SGX en el clúster. Además, existe otra anotación que se conoce con el nombre de *sgx.intel.com/epc*, la cual es utilizada por el *webhook* de admisión para ajustar dinámicamente el tamaño de la memoria caché en las páginas cifradas del *Enclave Page Cache* (EPC).
- ***SGX EPC memory registration***, permite registrar la memoria EPC disponible en cada nodo como un recurso de Kubernetes. Posteriormente, se instala una regla en *Node-Feature-Discovery* (NFD) para identificar la memoria SGX, y se configura NFD para registrarla automáticamente, de esta manera Kubernetes puede administrar de forma eficaz la memoria EPC en el clúster para las diversas cargas de trabajo que requieran SGX.

Lo segundo a destacar es que, si se desea que el clúster se pueda beneficiar de las propiedades de SGX será necesario realizar la instalación del *plugin* mencionado anteriormente *Intel SGX device plugin for Kubernetes*. El proceso de instalación requiere unos prerrequisitos. Estos son: (1) Un kernel Linux con los drivers SGX, disponibles en el kernel desde la versión ≥ 5.11 [22]; (2) El *hardware* SGX debe disponer de *Flexible Launch Control* (FLC), que permite al administrador de la plataforma escoger que *Launch Enclave* [23] (LE) (tipo especial de enclave) puede generar *launch tokens* o si los enclaves precisan o no de estos *tokens*; este tipo de *tokens* permiten garantizar que solo los enclaves autorizados y confiables puedan ser lanzados y ejecutados en un entorno de ejecución seguro; (3) Instalación de *cert-manager* [24], para gestionar certificados TLS en Kubernetes, y (4) Comprobar en los nodos si la asignación de recursos cuenta con el dispositivo SGX cargado.

Una de las particularidades de este *plugin* es que permite aceptar argumentos de línea de comandos como el número de contenedores por nodo para acceder a los dispositivos SGX, estos son: */dev/sgx_enclave* y */dev/sgx_provision*. De esta manera se consigue que las cargas de trabajo de Kubernetes empleen SGX, lo que resulta en un clúster confiable y

escalable.

Una vez conocidos los componentes y los prerrequisitos necesarios, se debe definir el proceso de instalación de este *plugin*. Actualmente existen tres maneras de realizar la instalación [21]: (1) Imágenes Docker previamente creadas, (2) Mediante el uso de un operador, y (3) Utilizando *kubectl*. Todas estas formas tienen en común una primera fase para efectuar el despliegue de NFD y posteriormente otra para el despliegue del *plugin*. De todas ellas, para el escenario actual se emplea la opción tres.

IV-B. Desarrollo de un servicio confiable

Para llevar a cabo la creación del servicio confidencial se utiliza el modelo de mantenimiento predictivo mencionado previamente en el escenario, donde una vez preprocesados los datos, se hace una normalización de estos. Para ello se transformarán los datos a un formato que permita introducirlos en una red *Long short-term memory* LSTM, para ulteriormente emplear un *autoencoder* que permita reconstruir los datos de entrada, siguiendo la estructura del modelo desarrollado en la publicación *Sensor Anomaly Detection* [25]. Finalmente, respecto al modelo, cabe añadir que este se ha entrenado con 100 *epochs*, y tras el entrenamiento y las pruebas de validación necesarias, se ha obtenido un modelo ML capaz de identificar fallos en el comportamiento de los sensores de vehículos. Para el desarrollo de este modelo se ha utilizado Python 3 [26] y la librería TensorFlow [27], además, el modelo ha sido exportado en formato *ProtoBuf (.pb)* para garantizar su funcionamiento tanto en TensorFlow, como gRPC [28], en el caso del primero se utiliza este formato para definir la estructura del modelo ML, mientras que en el segundo caso sirve para habilitar inferencias distribuidas sobre el modelo.

Como bien se ha indicado anteriormente, es necesario el uso de varias tecnologías para lograr este tipo de despliegues. Para esta casuística se empleará Gramine [29], la cual permite garantizar que las aplicaciones que no son mutuamente confiables no puedan inferir información entre ellas, garantizando así un aislamiento de seguridad comparable a la ejecución de máquinas virtuales aisladas. Este *library OS* [30] será utilizado para realizar operaciones de cifrado/descifrado sobre el modelo ML y sus variables aplicando la tecnología Intel SGX. A mayores, también se opta por el uso de la herramienta MarbleRun [31] para facilitar los procesos de despliegue, escalado y verificación del servicio TensorFlow. Gracias a MarbleRun, se podrá ejecutar de manera confiable el servicio con el modelo generado para la detección de fallos en sensores. Una vez desplegado este servicio confiable, los clientes podrán acceder a este e inferir datos de los sensores y recibir los resultados correspondientes, empleando el proceso de verificabilidad y un certificado. Se debe remarcar que estos datos inferidos viajan cifrados vía gRPC con TLS sobre el modelo detector de fallos que se encuentra en ejecución en el servidor TEE dentro del clúster confidencial como un contenedor que emplea Gramine; permitiendo así a los clientes que tengan permisos necesarios, poder recuperar resultados seguros sobre este modelo.

IV-C. Despliegue de un servicio confiable

Tras la obtención del modelo anterior, el siguiente paso será lograr que funcione dentro de un servidor TensorFlow que se ejecuta como un contenedor dentro del clúster confidencial SGX. Según la literatura actual, este procedimiento recibe el nombre de *Privacy Preserving Machine Learning with Intel SGX and TensorFlow Serving* [32], debido a que permite asegurar que la información sensible permanezca cifrada durante todo el proceso mediante el uso de la plataforma Intel SGX.

Seguidamente se exponen los pasos requeridos para lograr que el despliegue funcione, estos se muestran en la **figura 3**:

1. El modelo ML de detección de fallos se cifra y se sube cifrado a la nube (en este caso la nube sería el clúster del servidor TEE).
2. Se realiza el lanzamiento de MarbleRun utilizando un archivo de configuración, conocido como *manifest*, que establece la estructura y los elementos de la aplicación confidencial ML. Este procedimiento debe ser realizado por el administrador de la plataforma.
3. Seguidamente, el administrador debe desplegar la aplicación ML confidencial.
4. De manera autónoma, MarbleRun realiza los procesos de autenticación y arranque correspondientes a la plataforma.
5. Posteriormente, se verifica el despliegue del modelo vía MarbleRun. Se carga de manera segura la clave de cifrado AES generada con Gramine en la aplicación TensorFlow, utilizando la funcionalidad de gestión de secretos de MarbleRun.
6. La aplicación TensorFlow puede descifrar el modelo dentro del enclave mediante la clave proporcionada.

7–8 Finalmente, los clientes pueden verificar el despliegue usando MarbleRun, y a continuación se podrán conectar de manera segura al servicio sabiendo en todo momento que sus datos solamente pueden ser accedidos desde dentro del enclave.

Estos pasos anteriores reflejan el proceso de despliegue del servicio confidencial en Kubernetes utilizando un TEE (véase Código A). Durante la ejecución de estos pasos, ocurren una serie de comportamientos específicos a nivel interno, que son vitales para comprender adecuadamente el flujo de las operaciones efectuadas dentro del escenario. En la figura 4, se pueden ver estos comportamientos desglosados. Para poder abarcar un mayor conocimiento de lo que ocurre, a continuación, se desarrollarán algunos de los puntos anteriores en base a sus comportamientos complejos.

1.1 El proceso de obtención de clave y de cifrado del modelo ML se realiza a través de Gramine, en concreto, mediante la utilidad *gramine-sgx-pf-crypt*. Esta utilidad dispone de múltiples argumentos, y su comportamiento durante el despliegue es el siguiente: primero se ejecuta el argumento *gen-key* el cual permite generar una *wrap-key* empleando una fuente de números pseudoaleatorios segura; después, se emplea el argumento *encrypt*, que permite envolver el modelo y las variables con esta *wrap-key*. Posteriormente, tanto el modelo como las variables se cargarán en el clúster.

2.1 Antes de hacer el despliegue con MarbleRun, se necesita generar vía OpenSSL una clave (RSA) y un certificado (X.509) para el usuario. Una vez generados, se emplea la herramienta MarbleRun, primeramente para cargar

el archivo *manifest* que contiene la configuración del *Marble*/Servicio TensorFlow, así como el certificado del usuario, roles de este, etc. Después de haber cargado el *manifest* al clúster, se realiza la autenticación empleando la clave y el certificado mencionados al principio, para así poder cargar la clave generada con Gramine al clúster mediante la opción de secretos de MarbleRun. De esta manera, posteriormente tanto el modelo como las variables podrán ser descifradas dentro del clúster.

3.1 Para poder desplegar la aplicación, se debe construir un contenedor Docker. Conviene resaltar que este tiene varias fases en su construcción, estas son:

3.1.1 Instalación de las librerías necesarias como: SGX, TensorFlow, Edgeless RT (SDK y Runtime para operar fácilmente con Intel SGX) [33], y MarbleRun. Por otro lado, también se copiarán ciertos archivos al contenedor, como la plantilla de Gramine y un *script* de inicialización llamado *start.sh* el cual se describirá en el subapartado 3.1.3.

3.1.2 El archivo de plantilla Gramine (véase Código B) para TensorFlow, permite realizar una configuración minuciosa del LibOS de Gramine con SGX, pudiendo así especificar: el tamaño del enclave utilizado, hilos que puede emplear el procesador, archivos confiables, el nombre del servicio a ejecutar, etc.

3.1.3 La última fase consiste en la ejecución de *start.sh* que lleva a cabo tareas como: inicializar la dirección *url* de Intel SGX *Provisioning Certificate Caching Service* (PCCS) [34], el cual se usa para obtener certificados PCK [35] y otro material criptográfico, y finalmente ejecuta el servicio TensorFlow vía Gramine empleando el comando *gramine-sgx*.

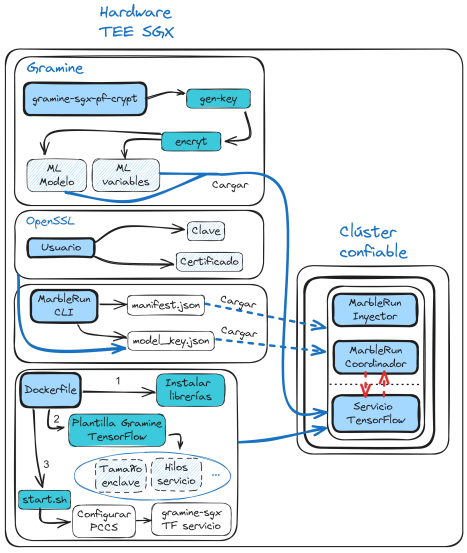

Figura 4. Comportamientos internos durante el despliegue.

Código B
ARCHIVO DE PLANTILLA GRAMINE.

```
loader.argv = ["tensorflow_model_server"]
sgx.enclave_size = "8G"
sgx.max_threads = 512
sgx.trusted_files = [
    "file:{{ gramine.libos }}",
    "file:{{ gramine.runtimedir() }}/",
    "file:tensorflow_model_server",
    ...
]
sgx.remote_attestation = "dcap" ...
```

Código A
ARCHIVO DE PLANTILLA KUBERNETES PARA TENSORFLOW.

```
...
serviceAccountName: tf-server
containers:
- image: ...
  resources:
    requests:
      memory: "12Gi"
      cpu: 8
      sgx.intel.com/epc: "8Gi"
      sgx.intel.com/enclave: 1
      sgx.intel.com/provision: 1
      ...
  env:
  - name: PCCS_URL
    value: {{ .Values.pccsURL }}
  - name: PCCS_USE_SECURE_CERT
    value: "{{ .Values.secureCert }}"
  - name: PCCS_DCAP
    value: {{ .Values.pccsDCAP }}
  - name: SGX_AESM_ADDR
    value: "1"
  - name: EDG_MARBLE_DNS_NAMES
    value: "grpc.tensorflow-serving.com"
  volumeMounts:
  - name: aesmd-socket
    mountPath: /var/run/aesmd
  - name: model-dir
  ...
```

V. RESULTADOS

Debido a que los (**TEEs**) están basados en hardware, las pruebas que se realizarán en este apartado han requerido del uso de un equipo con un procesador Intel Xeon Gold 5318S de 24 núcleos y 512 GB de RAM, el cual es compatible con SGXv2. Además, dentro de este se ha desplegado un clúster de Kubernetes configurado con 14 núcleos de CPU a 2.10 Ghz y 50 GiB de memoria RAM. Acto seguido, se ejecutará el modelo ML bajo un contenedor limitado cuyos recursos son 8 núcleos a 2.10 GHz, 12 GiB de memoria RAM, y un tamaño de enclave de 8 GiB.

Las pruebas efectuadas consisten en interactuar con el modelo detector de fallos. Por un lado, se llevarán a cabo dentro de un entorno de ejecución confiable, mientras que, por otro lado, se realizarán en un entorno estándar sin hacer uso del TEE. Se pretende observar como penaliza el rendimiento del servidor la utilización de las técnicas de seguridad implementadas y concluir si la penalización en el rendimiento puede o no suponer una merma significativa en el desempeño del servicio que haga inviable el uso de las técnicas descritas. Para lograr esto, se ha desarrollado un *script* que permite realizar *benchmarks* de carga. Estos *benchmarks* permiten inferir datos de un tamaño específico, seleccionando el número de clientes que interactúan y eligiendo la cantidad de inferencias por cliente (véase Código C).

Tras haber realizado los *benchmarks* correspondientes, se han obtenido dos tablas. Estas tablas contienen datos sobre el número de clientes y las predicciones efectuadas por estos (iteraciones) que tuvieron lugar en las pruebas. El objetivo es analizar los tiempos de latencia (expresados en milisegundos) para determinar cuanto tiempo tarda el modelo en procesar información en dos condiciones, un entorno seguro basado en TEEs y un entorno no seguro. Por un lado, tenemos la tabla **tabla II**, la cual contiene los datos de latencia utilizando un TEE. Por otro lado, obtenemos la tabla **tabla III**, que presenta los tiempos de latencia sin emplear un entorno confiable, es decir, sin ninguna medida de seguridad.

Código C
PSEUDOCÓDIGO PARA LAS PRUEBAS DE RENDIMIENTO.

```
1. Añadir argumentos:
  "URL gRPC"
  "Certificado TLS para gRPC"
  "Tamaño del lote"
  "Número de conexiones concurrentes"
  "Número de iteraciones"

2. BenchmarkEngine(argumentos)

3. Crear un bucle asincrónico.

4. Establecer eventos a utilizar.

5. Crear una lista de conexiones asincrónicas.

6. Para cada valor en el rango de conexiones
    concurrentes se añade una conexión asincróna.

7. Ejecutar todas las conexiones asincrónicas
    esperando a que estén completas.

8. Cerrar el bucle de conexiones asincrónicas.

9. Tiempo actual <- al final de la ejecución.

10. Calcular y mostrar <- (Conexiones concurrentes
    , tiempos e2e, latencia media, transferencias
     por segundo).
```

Tabla II
ANÁLISIS PERTINENTE ACERCA DEL IMPACTO DE LA LATENCIA UTILIZANDO UN TEE.

Num. clientes	Iteraciones		
	5 (ms)	15 (ms)	25 (ms)
1	7,082	1,959	4,712
20	14,724	10,054	9,231
50	20,853	19,096	24,792
100	50,151	48,362	49,902
200	100,256	96,207	102,048
300	138,770	142,619	158,115
400	194,711	161,808	168,470
450	250,081	191,557	183,281
500	302,489	205,392	208,757
600	339,786	315,411	386,833
700	447,126	317,887	438,891
800	531,647	428,751	444,681

Con base a los resultados obtenidos, lo primero que destaca es que el entorno confiable (**tabla II**) introduce una ligera sobrecarga que se traduce en una mayor latencia, lo que es esperable debido a las medidas de seguridad implementadas. Observamos que el procesamiento dentro de un TEE supera los 300ms a partir de las 5 iteraciones realizadas por 500 clientes. Por otro lado en la **tabla III** que refleja las latencias obtenidas fuera de un TEE, independientemente del número de iteraciones, la latencia en términos generales se encuentra ligeramente por debajo de los 300ms. Al comparar las latencias más altas de ambos escenarios, encontramos que en la **tabla II** se obtiene una latencia de $\approx$ 531ms, mientras que, en la **tabla III** se registra un máximo de $\approx$ 287ms, Esto implica un diferencia de alrededor de 244ms entre ambos escenarios. Además de la figura 5, podemos observar cómo la sobrecarga disminuye a medida que se conectan más clientes. Esto se debe a que el aumento en latencia debido al procesado dentro del TEE escala mejor cuando hay más clientes conectados. La razón entre las latencias observadas en la **tabla II** y la **tabla III** tiende a estabilizarse a medida que aumenta la demanda de procesamiento por parte de los clientes.

Tabla III
ANÁLISIS PERTINENTE ACERCA DEL IMPACTO DE LA LATENCIA SIN UTILIZAR UN TEE.

Num. clientes	Iteraciones		
	5 (ms)	15 (ms)	25 (ms)
1	2,049	1,436	1,163
20	2,873	3,004	3,110
50	7,341	6,336	5,828
100	13,595	13,024	11,192
200	39,864	23,543	21,899
300	59,068	38,540	35,070
400	86,309	56,676	55,571
450	115,871	64,341	59,429
500	125,805	76,746	66,062
600	187,190	104,278	84,600
700	217,089	129,156	106,321
800	287,341	152,953	126,758

VI. CONCLUSIONES Y LÍNEAS FUTURAS

El desarrollo de servicios confiables utilizando contenedores supone un avance significativo en la protección de datos sensibles en entornos *cloud*. La introducción de la tecnología Intel SGX en un entorno contenerizado con Kubernetes ofrece una solución tanto confiable como escalable y flexible, que permite mejorar la seguridad de los servicios en la nube actuales.

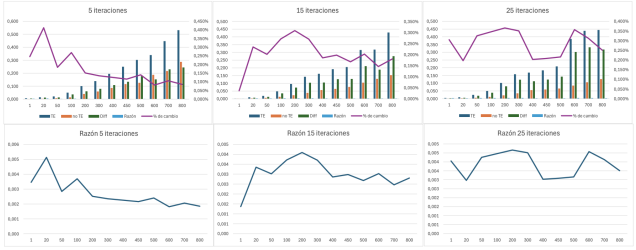

Figura 5. Comparativa resultados obtenidos.

El análisis de los datos obtenidos en las pruebas que hemos realizado, revelan que la adopción de un entorno de ejecución seguro (TEE) conlleva un incremento en el coste computacional. Este sobrecoste se estima en torno al 300 %, como se puede ver en la figura 5. A pesar de este incremento, nuestra valoración es que el sobrecoste es proporcionado y

razonable si lo comparamos con el ofrecido por el resto de tecnologías de procesamiento en el dominio cifrado [36]. La razón principal para esta conclusión es que operar dentro de un TEE nos permite proporcionar una capa adicional de seguridad para la aplicación, asegurando la integridad y la confidencialidad de los datos procesados. En un panorama donde las amenazas cibernéticas son cada vez más frecuentes y avanzadas, esta seguridad mejorada se convierte en un requisito indispensable.

Basándonos en nuestra investigación, hemos identificado varias direcciones para futuras investigaciones, que detallamos a continuación:

- Realizar un análisis detallado del comportamiento de Gramine para optimizar el rendimiento del servicio.
- Explorar otras implementaciones TEEs con contenedores [37] comparando el rendimiento ofrecido.
- Analizar y comparar el rendimiento y consumos de la tecnología utilizada con otras similares respecto al dominio del cifrado.
- Valorar el uso de imágenes de contenedores siempre cifradas. En este enfoque, las imágenes sólo se descifrarán durante el despliegue en el entorno seguro. La clave de descifrado necesaria será proporcionada por el desarrollador responsable.

Referencias

[1] TCS (Tata Consultancy Services). "Costs of Encrypted Computation: Why Fully homomorphic computations are Slow." [Online]. Available: https://www.tcs.com/content/dam/global-tcs/en/pdfs/insights/whitepapers/Fully-homomorphic-encryption-data-privacy.pdf

[2] M. Sabt, M. Achemlal, and A. Bouabdallah, "Trusted Execution Environment: What It is, and What It is Not," 2015 IEEE Trustcom/BigDataSE/ISPA, vol. 1, Aug. 2015, doi: https://doi.org/10,1109/trustcom,2015,357.

[3] A. Akram, A. Giannakou, V. Akella, J. Lowe-Power, and S. Peisert, "Performance Analysis of Scientific Computing Workloads on General Purpose TEEs." Accessed: Mar. 19, 2024. [Online]. Available: https://arch.cs.ucdavis.edu/assets/papers/ipdps21-hpc-tee-performance.pdf

[4] Confidential computing, How to process data securely on third-party infrastructure. [Online]. Available: https://content.edgeless.systems/hubfs/ConfidentialComputingWhitepaper.pdf. 2023.

[5] Cloud Native Computing Foundation, CNCF Annual Report 2022. [Online]. Available: https://www.cncf.io/reports/cncf-annual-report-2022/. Accessed: Mar. 20, 2024.

[6] Confidential Computing Consortium. "Confidential Computing: Hardware-Based Trusted Execution for Applications and Data" Confidential Computing Consortium. November 2022. [Online]. Available: https://confidentialcomputing.io/wp-content/uploads/sites/10/2023/03/CCC_outreach_whitepaper_updated_November_2022.pdf

[7] V. Costan and S. Devadas, "Intel SGX Explained," IACR Cryptology ePrint Archive, no. 2016/086, 2016. [Online]. Available: https://eprint.iacr.org/2016/086. Accessed on: Mar. 14, 2024.

[8] F. Kammel, M. Ylinen, and T. Feldman-Fitzthum, "Confidential Kubernetes: Use Confidential Virtual Machines and Enclaves to improve your cluster security," Jul. 26, 2023. https://kubernetes.io/blog/2023/07/06/confidential-kubernetes/ (accessed Mar. 19, 2024).

[9] B. Ngabonziza, D. Martin, A. Bailey, H. Cho, and S. Martin, "TrustZone Explained: Architectural Features and Use Cases," in 2016 IEEE 2nd International Conference on Collaboration and Internet Computing (CIC), Nov. 2016, pp. 1-8, doi: 10.1109/CIC.2016.0651.

[10] Arm Limited. (2021). Arm Confidential Compute Architecture. [Online]. Available: https://www.arm.com/architecture/security-features/arm-confidential-compute-architecture

[11] P.-C. Cheng, W. Ozga, E. Valdez, S. Ahmed, Z. Gu, H. Jamjoom, H. Franke, and J. Bottomley, "Intel TDX Demystified: A Top-Down Approach," arXiv preprint arXiv:2303.15540, Mar. 2023. [Online]. Available: https://arxiv.org/abs/2303,15540. Accessed on: Mar. 14, 2024.

[12] D. Kaplan, J. Powell, T. Woller, "Memory Encryption White Paper," AMD, Oct. 18, 2021. [Online]. Available: https://www.amd.com/content/dam/amd/en/documents/epyc-business-docs/white-papers/memory-encryption-white-paper.pdf. Accessed on: Mar. 14, 2024.

[13] D. Kaplan, "Protecting VM Register State with SEV-ES," AMD, Feb. 17, 2017. [Online]. Available: [AMD]. Accessed on: Mar. 14, 2024.

[14] "AMD Secure Encrypted Virtualization Solution Brief," AMD, Jan. 2020. [Online]. Available: https://www.amd.com/content/dam/amd/en/documents/epyc-business-docs/solution-briefs/amd-secure-encrypted-virtualization-solution-brief.pdf.Accessedon:Mar,14,2024.

[15] Confidential Computing Consortium, "Common Terminology for Confidential Computing," December 2022. [Online]. Available: https://confidentialcomputing.io/wp-content/uploads/sites/10/2023/03/Common-Terminology-for-Confidential-Computing.pdf

[16] Wang, Jinwen et al. "Interface-Based Side Channel Attack Against Intel SGX." ArXiv abs/1811.05378 (2018): n. pag.

[17] J. Ménétrey et al., Attestation mechanisms for trusted execution environments demystified, Distributed Applications and Interoperable Systems, Cham: Springer International Publishing, 2022, pp. 95-113.

[18] Rob Nertney, Çonfidential Compute on NVIDIA Hopper H100", Nvidia, July 25, 2023. [Online]. Available: https://images.nvidia.com/aem-dam/en-zz/Solutions/data-center/HCC-Whitepaper-v1,0.pdf

[19] A. Mogage, R. Pires, V. Crăciun, E. Onica and P. Felber, "Supply Chain Malware Targets SGX: Take Care of what you Sign,"2019 38th Symposium on Reliable Distributed Systems (SRDS), Lyon, France, 2019, pp. 52-528, doi: 10.1109/SRDS47363.2019.00016.

[20] Hai Qiu, Jay Lee, Jing Lin, "Wavelet Filter-based Weak Signature Detection Method and its Application on Roller Bearing Prognostics,"Journal of Sound and Vibration, Vol. 289, pp. 1066-1090, February, 2006, doi: 10.1016/j.jsv.2005.03.007.

[21] Intel Device Plugins for Kubernetes, Overview. [Online]. Available: https://intel.github.io/intel-device-plugins-for-kubernetes/README.html. 2024.

[22] Intel® SGX Software Installation Guide, For Linux* OS. [Online]. Available: https://download,01.org/intel-sgx/latest/linux-latest/docs/Intel_SGX_SW_Installation_Guide_for_Linux.pdf. 2024.

[23] Intel(R) SGX Reference Launch Enclave. [Online]. Available: https://github.com/intel/linux-sgx/blob/master/psw/ae/ref_le/ref_le.md. 2023.

[24] Kubernetes - Installing with regular manifests, cert-manager. [Online]. Available: https://cert-manager.io/v1,1-docs/installation/kubernetes/. 2024.

[25] Sensor anomaly detection, Kaggle.com. [Online]. Available: https://www.kaggle.com/code/rkuo2000/sensor-anomaly-detection/notebook. 2020.

[26] Welcome to Python.org. [Online]. Available: https://www.python.org/. 2024.

[27] TensorFlow. [Online]. Available: https://www.tensorflow.org/. 2024.

[28] GRPC. [Online]. Available: https://grpc.io/. 2024.

[29] Gramine, Gramineproject.io. [Online]. Available: https://gramineproject.io/. 2024.

[30] C.-C. Tsai, D. E. Porter, y M. Vij, Graphene-SGX: a practical library OS for unmodified applications on SGX, Proceedings of the 2017 USENIX Conference on Usenix Annual Technical Conference, 2017, pp. 645-658, doi: 10.5555/3154690.3154752.

[31] MarbleRun: the service mesh for confidential computing, Edgeless.systems. [Online]. Available: https://www.edgeless.systems/products/marblerun/. 2024.

[32] Reference Architecture for Privacy Preserving Machine Learning with Intel® SGX and TensorFlow* Serving. [Online]. Available: https://www.intel.com/content/www/us/en/developer/articles/technical/privacy-preserving-ml-with-sgx-and-tensorflow.html. 2024.

[33] Edgeless RT. [Online]. Available: https://github.com/edgelesssys/edgelessrt. 2024.

[34] Provisioning Certificate Caching Service (PCCS). [Online]. Available: https://github.com/intel/SGXDataCenterAttestationPrimitives/blob/master/QuoteGeneration/pccs/README.md. 2024.

[35] Remote Attestation for Multi-Package Platforms using Intel® SGX Datacenter Attestation Primitives (DCAP). [Online]. Available: https://download,01.org/intel-sgx/latest/dcap-latest/linux/docs/Intel_SGX_DCAP_Multipackage_SW.pdf. 2021.

[36] Performance Impact Analysis of Homomorphic Encryption: A Case Study Using Linear Regression as an Example, Researchgate.net. [Online]. Available: https://www.researchgate.net/publication/375473628_Performance_Impact_Analysis_of_Homomorphic_Encryption_A_Case_Study_Using_Linear_Regression_as_an_Example. 2023.

[37] CoCo, Confidential containers. [Online]. Available: https://confidentialcontainers.org/. 2024.