Análisis de los servicios criptográficos en la nube pública

Juan Enrique Gómez Pérez

Máster Universitario en Seguridad de las Tecnologías de la Información y las Comunicaciones

Sistemas de autenticación y autorización

Tutor: Pau del Canto Rodrigo

Enero 2022

Agradecimiento

Fundamentalmente a mi familia que ha aguantado mis largas horas de trabajo, viajes, y además todas las horas robadas en este Master, así como este trabajo. Y a mi madre que me enseñó que todo lo que me proponga es posible, aquí una muestra.

FICHA DEL TRABAJO FINAL

Título del trabajo:	*Análisis de los servicios criptográficos en la nube pública*
Nombre del autor:	*Juan Enrique Gómez Pérez*
Nombre del consultor/a:	*Pau del Canto Rodrigo*
Fecha de entrega (mm/aaaa):	01/2022
Titulación:	*Máster Universitario en Seguridad de las Tecnologías de la Información y de las Comunicaciones*
Área del Trabajo Final:	*Sistemas de autenticación y autorización*
Idioma del trabajo:	Castellano
Palabras clave	*PKI HSM Criptografía*

Resumen del trabajo (máximo 250 palabras): *Con la finalidad, contexto de aplicación, metodología, resultados y conclusiones del trabajo.*

La nube pública nace en 2006 con Amazon Web Services y ha experimentado una enorme adopción con la migración de prácticamente cualquier tipo de servicio o datos hacía ella. La criptografía a día de hoy juega un papel crítico en la seguridad y autenticidad de las actividades que realizamos y sin duda es un servicio imprescindible para mover estas cargas de trabajo. Entender las posibilidades que nos ofrece la nube, como operar y manejar estos sistemas, como funcionan los protocolos de protección de claves, o si los procesos nos permiten la interoperabilidad entre diferentes sistemas criptográficos, son una serie de cuestiones relevantes que analizaremos en este trabajo. Para ello desplegamos una PKI en la nube pública y analizaremos estos aspectos basándonos en diferentes casos de uso más comunes en la industria: protección de los datos en tránsito (certificados SSL/TLS) y protección de los datos en reposo (encriptación de objetos de datos o volúmenes).

El objetivo es demostrar la viabilidad de desplegar una PKI completamente funcional, demostrar la interoperabilidad que existe entre sistemas tradicionales y sistemas en la nube en la gestión y protocolos de claves criptográficas, y finalmente analizar la facilidad de integración de estos sistemas utilizando protocolos estándar como ACME.

Abstract (in English, 250 words or less):

Public cloud was born in 2006 with Amazon Web Services, and during all these years it has been broadly adopted. Any type of workload or data has been suitable to be migrated into the public cloud. Today, cryptography is a critical piece for all those workloads moving into the public cloud, bringing security and authenticity to them. Understanding the services available on the public cloud, how to operate and manage these systems, how

the key protection protocols works, or if these processes allow us to interoperate between different crypto systems are relevant questions that we will analyze during this document.

To achieve these objectives, we will deploy a PKI on a Public Cloud vendor and we will go through all these aspects, trying to identify some relevant use cases which are relevant for the industry. We will focus on how to protect with a PKI data in transit (SSL/TLS certificates).

The objective of this document, is to show the viability of deploying a completely functional PKI using cloud native services, demonstrate interoperability with other crypto services using traditional devices, to finally build a PoC to use this PKI at scale using a protocol like ACME.

Índice

Lista de figuras

1. Introducción

1.1. Contexto y justificación del Trabajo

Los servicios de nube pública cada día cuentan con una mayor penetración en el mercado tanto empresarial como también en sector público. Desde la aparición de Amazon Web Services en el año 2006 la adopción de estos servicios en la nube no ha hecho más que crecer año a año a ritmos del 38% con un mercado actual de 312 billones de dólares (IDC)

Esta gran adopción hace que la ciberseguridad desde hace unos años haya sido uno de los aspectos principales, en los que más foco han puesto estos proveedores de servicios en la nube en general, pero donde el esfuerzo ha sido especialmente destacable es en la protección de los datos de sus clientes.

Los sistemas criptográficos juegan un papel vital en la protección de los datos, tanto si estos se encuentran en tránsito (viajando entre dos sistemas) como si se encuentran almacenados en un sistema permanente. Garantizar que estos datos sólo son accesibles en claro por los destinatarios que se pretendía inicialmente es crítico, y es algo que los sistemas de información criptográfica garantizan.

La nube basa gran parte de su propuesta de valor en lo que se denominan servicios gestionados, que tienen un reflejo en un modelo de responsabilidad compartida entre el usuario de los servicios y el proveedor de servicios. Este modelo de responsabilidad compartida implica que el proveedor de nube asume la responsabilidad de parte del *stack* tecnológico, y otra parte es gestionada por el cliente. Este aspecto es relevante en un mundo como el de los servicios criptográficos, en los que en muchas ocasiones existen ceremonias de intercambio de claves, que implican acceso físico a dispositivos y que no es posible en un servicio gestionado en la nube.

Este nuevo modelo de servicio plantea algunas cuestiones que deben ser contestadas en este trabajo de final de máster, ¿son equivalentes los sistemas criptográficos tradicionales y en la nube? ¿Puedo llevar a cabo los mismos procedimientos y operaciones?, o ¿los protocolos existentes son intercambiables?

Este trabajo de manera teórica y práctica analizará un caso de uso relevante para los clientes empresariales y fundamental para cliente regulados (sector bancario, asegurador, sanitario, …) que es la viabilidad de utilizar sistemas criptográficos en la nube que me permitan mantener mis operaciones y controles actuales, así como ceremonias. Pero, sobre todo, que ocurre con mis datos y mecanismos de protección en caso que el sistema criptográfico del proveedor de nube no esté disponible, ¿Puedo recuperar mis datos? ¿Puedo seguir asegurando el nivel de seguridad adecuado?

1.2. Objetivos del Trabajo

El trabajo plantea el análisis de los sistemas criptográficos que se ofrecen en proveedores de servicios de nube pública, como Amazon Web Services, los cuales funcionan como un servicio gestionado.

Durante este trabajo nos centraremos en primer lugar qué tipo de servicios son los que se ofrecen relacionados con la criptografía en general, y con la posibilidad de construir una PKI en particular.

Entendido la oferta existente, así como sus funcionalidades nos centraremos en entender sus limitaciones o condiciones de funcionamiento comparado con sistemas estándar de mercado como pueden ser sistemas HSM, así como los protocolos que se permiten en

estos, ya que en muchas ocasiones por ejemplo las ceremonias de intercambio de claves son un requisito imprescindible para ciertas funciones de una PKI.

Finalmente, de forma práctica, desplegaremos una prueba de concepto, utilizando estos servicios gestionados. El objetivo teórico será construir una PKI completamente funcional, y que podamos utilizar (entendiendo las potenciales limitaciones) con un protocolo como ACME para gestionar el ciclo de vida de los certificados y su uso para proteger los datos en tránsito. El uso práctico de esta PKI se centrará en su utilización en un entorno de microservicios utilizando un orquestador como *Kubernetes* y sistemas de gestión de mallas de microservicios como *Istio*.

Estos objetivos los dividiremos en tres bloques:

Análisis:

- Análisis de la oferta de servicios de los 3 proveedores de nube pública principales (Microsoft Azure, Google Cloud y Amazon Web Services)
- Descripción de sus funcionalidades y viabilidad de casos de uso
- Revisión de los protocolos de intercambio de claves disponibles
- Algoritmos soportados y ciclo de vida de estos servicios
- Compatibilidad general con sistemas estándar de mercado

Construcción del prototipo

- Selección de servicio gestionado a utilizar del *vendor* de nube pública
- Despliegue de una PKI en nube pública
- Integración de este servicio con el protocolo ACME
- Utilización en diferentes arquitecturas de esta PKI para la emisión, instalación y renovación de certificados SSL/TLS

Conclusiones

- Alcanzar una visión clara del estado del arte de los servicios de criptografía gestionados en nube pública
- Demostrar la viabilidad de usar estándares en la industria como ACME con estos servicios de nube pública
- Analizar la compatibilidad entre los sistemas tradicionales y de nube pública

1.3. Enfoque y método seguido

El presente trabajo tiene un reto técnico, que es la construcción de una PKI sobre infraestructura y servicios gestionados de nube pública que pueda ser utilizada de manera lo más automática posible en la comunicación de diferentes servicios de forma encriptada (comunicación SSL/TLS).

Alrededor de este modelo técnico, se explorarán diferentes aspectos de cara a su puesta en marcha.

Arrancaremos con la revisión y análisis de los diferentes servicios que ofrecen los proveedores de nube pública relacionados con la criptografía de clave pública, de esta manera entender que casos de uso son viable o no (¿Puedo montar una PKI completa? ¿Puedo utilizar esta PKI para qué propósitos? etcétera).

Este entendimiento llevará a tomar algunas de las primeras decisiones técnicas de este trabajo, de cara a seleccionar los servicios adecuados o que más se puedan ajustar a las necesidades. A partir de aquí se lanzarán los primeros trabajos del prototipo técnico en 3 frentes diferentes: construcción de la PKI y los servicios que esta debe proporcionar,

capacidad de integración con un protocolo de utilización de la PKI como ACME, y finalmente utilización en un caso práctico para la encriptación del tráfico en tránsito entre diferentes sistemas.

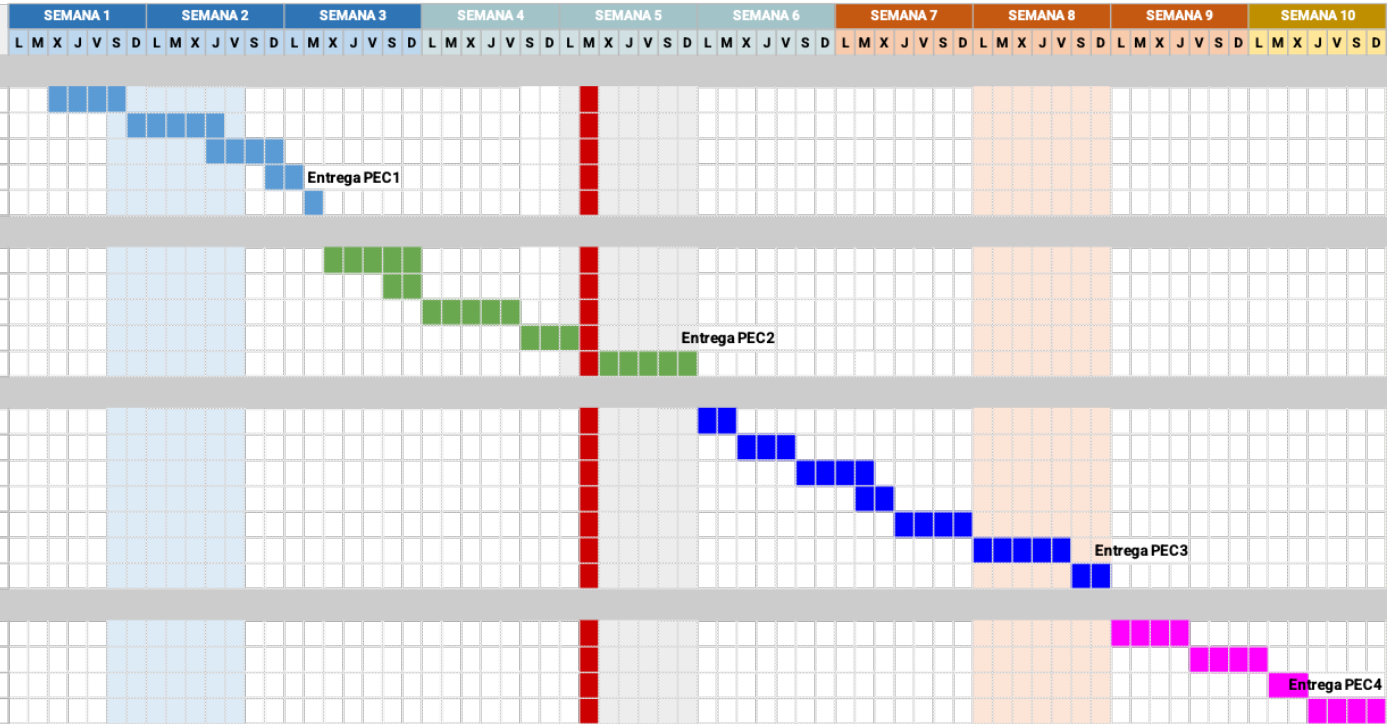
Esta práctica técnica, derivará en algunas lecciones aprendidas, que nos ayudarán a acercarnos a las conclusiones del trabajo. Estas conclusiones validarán la viabilidad de disponer de una PKI usando infraestructura de nube pública en primer lugar. En segundo lugar, tendremos una foto clara del estado del arte y que casos de uso son susceptibles de utilizar este tipo de PKI. Y finalmente uno de los objetivos principales del trabajo es entender la interoperabilidad entre entornos de nube pública y tradicionales, de manera que podremos extraer las conclusiones necesarias sobre si esta arquitectura es compatible o replicable con una similar construida con servicios tradicionales.

Estas conclusiones se completarán con un juicio de valor que los servicios de nube pública gestionados aportan en un entorno como el descrito.

La estrategia descrita tiene un fuerte componente técnico y de análisis funcional, ya que es muy importante profundizar en aspectos técnicos de las soluciones. En general en la nube pública los servicios “parecen” idénticos a otros servicios, pero su naturaleza de servicios gestionados hace que determinadas funcionalidades no estén disponibles o accesibles por el consumidor. Por este motivo, es muy importante que nuestra aproximación incluya este componente de estudio, análisis del caso de uso, validación técnica mediante una prueba de concepto, y entonces obtención de conclusiones objetivas de los resultados.

1.4. Planificación del Trabajo

1	**Análisis e inicio del proyecto**		
1.1	Analizar y decidir tema de proyecto	15/09/21	18/09/21
1.2	Acordar los objetivos	19/09/21	23/09/21
1.2	Escribir el abstract/objetivos/metodología	23/09/21	26/09/21
1.3	Definir tareas	26/09/21	27/09/21
1.4	Construir cronograma	28/09/21	28/09/21
2	**Fase de Analisis**		
2.1	Analizar la oferta de los vendors de nube	29/09/21	03/10/21
2.2	Identificar casos de uso	02/10/21	03/10/21
2.3	Analisis funcional de los casos de uso	04/10/21	08/10/21
2.4	Revisión de protocolos y algoritmos	09/10/21	11/10/21
2.5	Compatibilidad estandares de mercado	13/10/21	17/10/21
3	**Arranque del prototipo**		
3.1	Elección servicio gestionado para la PKI	18/10/21	19/10/21
3.2	Despiegue e la PKI	20/10/21	22/10/21
3.3	Despliegue y configuración de ACME	23/10/21	26/10/21
3.4	Diseño de arquitecturas a desplegar	26/10/21	27/10/21
3.5	Automatizació y despliegue	28/10/21	31/10/21
3.6	Integración con ACME con la plataforma	01/11/21	05/11/21
3.7	Pruebas de la plataforma	06/11/21	07/11/21
4	**Producción y entrega del TFM**		
4.1	Inicio conclusiones y estado del arte	08/11/21	11/11/21
4.2	Conclusiones del prototipo y su viabilidad	12/11/21	15/11/21
4.3	Analisis compatibilidad entre sistemas	16/11/21	17/11/21
4.4	Conclusiones	18/11/21	21/11/21



1.5. Breve sumario de productos obtenidos

Durante el presente trabajo hemos realizado tres grandes tareas. En primer lugar, hemos realizado un análisis de la oferta de servicios de los proveedores de nube púbica, centrándonos en los tres más relevantes en la actualidad: Amazon Web Services, Microsoft Azure y Google Cloud. Hemos realizado una visión del estado del arte de los tres orientado a los servicios de criptografía existentes, para completar con una justificación de la relevancia e importancia a día de hoy de los servicios de nube pública, lo que justifica totalmente este trabajo.

A continuación, hemos hecho un análisis detallado de los servicios, funcionalidades y capacidades existentes de los servicios especializados de criptografía para la nube pública, con especial foco en los denominados módulos de hardware de seguridad o HSM tanto como dispositivos dedicados como servicios gestionados. Esta distinción es muy relevante ya que en cada caso los denominados modelos de responsabilidad compartida han mostrado que nuestra gestión y control de los diferentes dispositivos y servicios tiene un alcance distinto.

Por último, hemos querido analizar la complejidad y la interoperabilidad de construir una solución de PKI utilizando estos servicios de nube pública. Esto nos ha llevado a disponer de una PKI totalmente funcional usando el software EJBCA en su versión de comunidad y respaldado por una solución HSM proporcionada en la nube como un servicio gestionado denominada AWS CloudHSM. Algunos aspectos interesantes que se han obtenido es la arquitectura centralizad de esta PKI que permite su reutilización en otras cargas de trabajo residentes en AWS de manera segura utilizando el servicio nativo VPC EndPoint y sin limitar la integración con terceros fuera de AWS.

2. Nube pública

2.1. Introducción a la nube pública

La conocida como nube pública trata de la distribución de servicios de TI bajo demanda a través de internet bajo un modelo de pago por uso de estos servicios. Los servicios TI consumidos en este modelo funcionan como podría ser un proveedor de electricidad, si enciendes la luz el servicio dispone de un contador y se facturará por periodos de tiempo predeterminados, y si por el contrario se apaga la luz, la electricidad dejará de fluir y no pagaremos por ello. La nube pública tiene un modelo de funcionamiento similar, permite en cualquier momento disponer de los recursos de TI que necesitamos, de manera ubicua y prácticamente ilimitada, pero con esta interesante característica del pago por uso.

Este modelo permite que, en vez de comprar activos de TI, y tener que realizar complejos planes de inversión como hasta ahora, podemos alinear el gasto en TI a las necesidades concretas en cada momento de los proyectos. Además, la disposición inmediata de los recursos de TI a la escala necesaria lo hacen altamente atractivo, evitando así las grandes inversiones económicas en activos que teníamos que realizar hasta la fecha sin una garantía de éxito para nuestros proyectos.

En la actualidad los principales fabricantes del mundo ofrecen servicios de nube pública con diferentes particularidades, pero todos ellos para ser considerados nube pública comparten estas características descritas en la introducción. Los principales proveedores a fecha de este trabajo son Amazon Web Services, Microsoft y Google por orden de penetración en el mercado.

En la actualidad el uso de la nube pública está ampliamente extendido en organizaciones de todo tipo, desde pequeñas *startups* hasta grandes compañías financieras, pasando por entidades educativas o de sector público.

Las principales ventajas que podemos enumerar de la nube pública son las siguientes:

- Agilidad: ofrece acceso a una amplia gama de servicios de TI de manera inmediata. Los servicios TI ofrecen una amplia variedad de estos, desde servicios más sencillos como puede ser pura computación en forma de máquinas virtuales, o almacenamiento, a servicios de plataforma como bases de datos gestionadas, o servicios más avanzados como inteligencia artificial con modelos ya entrenados para casos de uso concretos. Todo esto disponible de manera prácticamente inmediata mediante el uso de los servicios de nube pública.
- Elasticidad: Como hemos comentado los servicios son prácticamente infinitos, lo que nos permitirá en cada momento consumir tantos recursos de cada servicio como nos sea necesario, de manera que en el momento que no los necesitemos podremos apagarlos o eliminarlos. Introduciendo de esta manera una gran eficiencia de costes en los modelos, así como mejorando en general los resultados de nuestros proyectos al disponer siempre de la cantidad de recursos necesaria.
- Ahorro de costes: Esa elasticidad y agilidad, mezclada con los modelos de pago por uso, nos permitirán ahorrar costes, ya que evitarán las grandes inversiones iniciales en servicios de TI, y nos permitirán una mayor granularidad desplegando en cada momento los recursos que necesitemos, y eliminándolos cuando ya no los necesitemos. De esta manera solo pagaremos por lo que realmente estemos consumiendo en cada momento, y no pagaremos por recursos que tenemos disponibles por si fuera necesario su uso.
- Alcance global: los servicios de nube pública ofrecen lo que se denominan regiones, que son ubicaciones en diferentes partes del mundo, en la que se pueden desplegar

estos recursos TI. De esta manera desde una única ubicación podremos desplegar cualquiera de los servicios TI en cualquier ubicación en el globo en la que nuestro proveedor de nube pública proporcione servicio.

2.2. Tipos de servicios en la nube

La nube pública como ya hemos descrito se centra en la provisión de servicios de TI de manera ágil, elástica, consumibles por internet y en un formato de pago por uso. Los proveedores de nube pública pueden ofrecernos servicios de TI más o menos elaborados y esto nos permite dividirlos en tres grupos diferentes:

- Infraestructura como servicio (IaaS): Los servicios IaaS ofrecen la capa más básica de servicios de TI, que generalmente es computación en forma de máquinas virtuales, almacenamiento de objetos, ficheros o bloques y red. El proveedor de servicios se hará responsable de todo lo necesario para prestar estos servicios a la escala que se demande, centro de datos, seguridad, red, sistemas de gestión, mantenimiento, etc. de manera que el cliente podrá obtener a la escala requerida estos servicios.
- Plataforma como servicio (PaaS): Los servicios PaaS incluyen los servicios IaaS, pero ofrecen alguna funcionalidad orientada a la ejecución de aplicaciones o bases de datos. De esta manera servicios como el mantenimiento del sistema operativo, o incluso de los motores del servidor de aplicaciones o bases de datos son parte de la responsabilidad del proveedor de servicios de nube, ofreciendo en este servicio PaaS directamente el servicio de ejecución de aplicaciones o bases de datos entre otros.
- Software como Servicio: Es la capa más elaborada de servicio en la nube, que en general ofrecen una funcionalidad completa para el usuario final. Servicios como correo electrónico, CRM, contact centers, son servicios SaaS en las que el proveedor de servicios de nube se responsabiliza de toda la pila necesaria para su ejecución.

2.3. Servicios criptográficos de nube pública

Como parte de los servicios gestionados ofrecidos por los proveedores de nube pública, todos ellos ofrecen dos tipos de servicios criptográficos. El primero de ellos son las capacidades criptográficas de los propios servicios, son en general una función que forma parte del propio servicio, dotando a este de esas capacidades de encriptación de manera nativa sin que el usuario deba hacer nada especial. En general en este caso de uso estas capacidades están dirigidas bien a la protección de los datos (encriptación) en reposo, a la protección de los datos cuando estos están siendo transmitidos entre dos sistemas, o en tercer lugar su uso para los procesos de identificación (como mTLS o peticiones de APIs firmadas) entre sistemas utilizando técnicas criptográficas.

En segundo lugar, encontramos servicios dedicados en exclusiva a prestar funciones criptográficas, que van desde la generación, almacenamiento y gestión del ciclo de vida de las claves criptográficas, hasta la capacidad de ejecutar algoritmos criptográficos de clave simétrica, asimétrica, o funciones hash o de firma. Este segundo tipo de servicios pueden tener dos funciones diferentes, una primera función destinada a ofrecer soporte a las capacidades criptográficas del resto de servicios del proveedor de nube pública, y una segunda función que es ser consumida directamente por los usuarios de la misma manera que harían con servicios de tipo HSM en su propio centro de datos.

A continuación, revisaremos la oferta de servicios propuesta por parte de los tres proveedores de nube pública y como estas se integran con el resto de servicios.

2.3.1. Amazon Web Services

AWS ofrece dos grupos de servicios según su propia clasificación, un primer grupo denominado servicios criptográficos y un segundo grupo denominado servicios de PKI.

2.3.1.1. Servicios Criptográficos

2.3.1.1.1. AWS Cloud HSM

AWS CloudHSM es un módulo hardware físico criptográfico de seguridad disponible en la nube que le permite generar sus propias claves de cifrado, así como gestionar su ciclo de vida, almacenarlas de manera segura, como ejecutar operaciones criptográficas de clave simétrica o asimétrica.

AWS CloudHSM es un hardware físico totalmente dedicado para el cliente. Este es uno de los pocos servicios que no utiliza servicios de virtualización para compartir el hardware físico subyacente. Esto permite a AWS CloudHSM disponer de certificaciones como FIPS 140-2 nivel 3.

AWS CloudHSM ofrece la capacidad de integrar las aplicaciones con API estándares de mercado como pueden ser PKCS#11, Java Cryptography Extensions y bibliotecas de Microsoft CryptoNG (CNG).

AWS CloudHSM se integra de manera nativa con los servicios de nube pública de AWS de manera que podemos utilizarlo como un servicio más dentro de nuestros servicios desplegados en AWS, a la vez que su arquitectura permite su consumo desde otras ubicaciones (por ejemplo, desde nuestro centro de datos on-premise) de manera segura. AWS CloudHSM puede actuar mediante el uso de sus librerías como un almacén de claves seguro, y hardware para realizar las operaciones criptográficas para nuestras aplicaciones, servicios, o incluso para servicios nativos de criptografía de AWS que requieren un nivel de gestión de claves como FIPS 140-2 nivel 3.

Todas estas características, a pesar de ser un Hardware dedicado, mantienen las características básicas descritas en la. Introducción de este documento de la nube pública como la disponibilidad inmediata, la escalabilidad mediante la capacidad de añadir nuevos dispositivos como parte del clúster, o pago por uso.

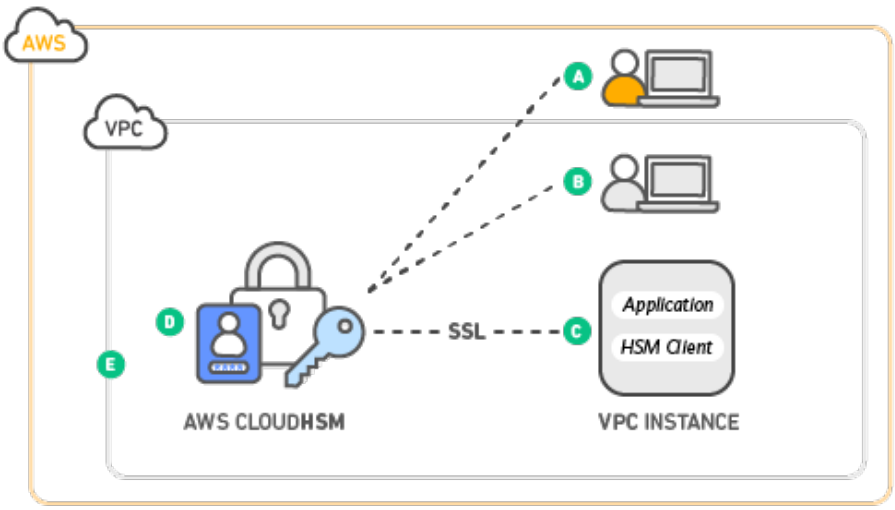


Ilustración 1 - Modelo despliegue CloudHSM

Como se ve en la imagen, AWS CloudHSM se despliega dentro de la nube privada del cliente (VPC) y es accesible de manera local permitiendo así una latencia ultra-baja en las comunicaciones entre las aplicaciones y los dispositivos. AWS no tiene acceso a las claves de AWS CloudHSM las cuales son gestionadas por el cliente.

Siguiendo el paradigma del modelo de responsabilidad compartida AWS mantiene algunas de las operaciones necesarias en la nube como son:

- Despliegue y mantenimiento del hardware físico
- Ejecución de las copias de seguridad de los dispositivo y salvaguarda de los mismos, con claves especificas de fabricante y de copia de seguridad
- Recolección de logs hacía el servicio de logs de AWS CloudWatch Logs
- Conexión a la red privada del cliente

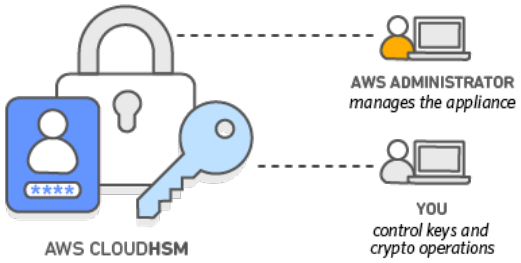


Ilustración 2 - Modelo de responsabilidad CloudHSM

El resto de las funciones son gestionadas por el cliente, inicialización de los módulos criptográficos, gestión de claves, ciclo de vida, control de accesos, operaciones criptográficas, integración con otros servicios nativos de AWS o de terceros.

El servicio de AWS CloudHSM ofrece soporte para tres casos de uso principales:

- Procesamiento SSL/TLS y off-loading para servidores web/aplicaciones
- Protección de claves privadas para autoridades de certificación
- Soporte para Transparent Data Encryption para bases de datos Oracle

2.3.1.1.2. AWS Key Management Service (AWS KMS)

AWS KMS es el segundo servicio con capacidades equivalentes a las de un módulo de hardware criptográfico tipo HSM. AWS KMS está soportado por una infraestructura de dispositivos HSM físicos, los cuales ofrecen características similares a un HSM tradicional, pero en esta ocasión lo que obtenemos es una partición de estos dispositivos físicos que se consumen como un servicio nativo de AWS, es decir vía llamadas a la API de AWS o bien usando el SDK especifico de AWS.

AWS KMS es un servicio diseñado para escalar a cualquier tamaño que el usuario pueda demandar, ya que se va a convertir en la pieza criptográfica fundamental de su infraestructura para todos los procesos de generación de claves, gestión del ciclo de vida de estas y utilización de sus claves para encriptar en reposo (almacenamiento de los datos). En los últimos tiempos ha puesto a disposición la capacidad de gestionar claves no solo simétricas, si no también asimétricas, pero con unas funcionalidades bastante limitadas en general.

AWS KMS ofrece un nivel de seguridad certificado FIPS 140-2 nivel 2. AWS KMS es un servicio totalmente administrado, en el que la responsabilidad del usuario se centra en los servicios criptográficos, generación de claves, definición del ciclo de vida, consumo o integración de los servicios de criptografía con otros servicios. El resto de la administración del servicio es responsabilidad de AWS, provisión, mantenimiento, escalabilidad, programas de conformidad, y auditoría.

AWS KMS se basa principalmente en el uso de claves simétricas ya que su misión es la de proporcionar servicios de criptografía a los servicios nativos de AWS. A este respecto el modelo funciona con una clave principal (Customer Master Key) la cual puede ser de tres tipos diferentes:

- AWS Owned Key: Este tipo de CMK existe cuando uno de los servicios de AWS que estamos utilizando directamente ha gestionado la creación de las claves con el servicio de AWS KMS y nosotros no tenemos ninguna capacidad de gestión sobre la clave, incluso sobre su rotación.
- AWS Managed Key: En este caso es el cliente quien le pide a AWS KMS la creación de una clave que es generada y custodiada dentro de AWS KMS y no es exportable. Esta clave puede ser rotada como máximo cada 1095 días (3 años). El cliente define las políticas de acceso a la clave y quien la puede utilizar.
- Customer Managed Key: En este caso el cliente puede genera su clave simétrica fuera del AWS KMS e importarla mediante mecanismos de Wrapping en AWS KMS para utilizarla como la CMK del servicio. En ese caso al haberse generado la clave fuera de AWS KMS no es posible la rotación de esta clave y será responsabilidad del cliente gestionar su rotación de manera manual.

La clave CMK se utiliza para encriptar la clave de datos que genera AWS KMS a petición de cualquier aplicación o servicio. Esta clave de datos se almacena encriptada con la CMK al lado de los datos de manera que posteriormente para recuperar estos datos, se le solicitará a KMS una versión en claro de la clave de datos que este desencriptará con la CMK que tiene gestionada.

Es interesante describir como funciona AWS KMS, ya que según vemos puede parecer equivalente a un HSM pero gestionado, y tiene algunas particularidades que lo hace muy útil dentro del ecosistema AWS pero complejo de utilizar fuera de este. El motivo está relacionado con el mecanismo que tiene de encriptación y la aplicación del concepto de "contexto".

AWS KMS cuando realiza operaciones de desencriptación no solo se basa en el texto cifrado sino también utiliza en sus operaciones criptográficas un elemento denominado contexto. El contexto es información que se almacena junto con el texto cifrado y que contiene información en forma de pares de clave valor. Este contexto tiene que ser proporcionado de manera adecuada cuando solicitando una operación de desencriptación a KMS. En general los servicios de AWS utilizan este contexto para ser capaces de solicitar únicamente ellos la desencriptación de este texto cifrado. Por ejemplo, la siguiente imagen muestra como el servicio de almacenamiento de AWS, EBS, solicita la desencriptación de un volumen de datos proporcionando un contexto especifico para este volumen:

```
"encryptionContext": {
  "aws:ebs:id": "vol-0cfb133e847d28be9"
}
```

2.3.1.1.3. Servicios de PKI

AWS ofrece un servicio de infraestructura de clave publica (PKI) tanto como autoridad de certificación reconocida, como de clave privada. El servicio se denomina AWS Certificate Manager (ACM) y es un servicio totalmente gestionado, que ofrece muchas de las funcionalidades necesarias para operar una PKI de carácter público o privado.

Una PKI es un servicio que está compuesto por un hardware que ofrece servicios criptográficos como los ya descritos, software, políticas, documentación y procedimientos. El objetivo de una PKI es la emisión de certificados, su gestión, distribución, uso, almacenamiento, validación y revocación llegado el caso. Una PKI se basa en los servicios de criptografía hardware los cuales garantizan una operación segura de todos los elementos como claves de la autoridad certificadora, claves privadas emitidas para los diferentes certificados tanto públicos como privados.

AWS Certificate Manager se ofrece como servicio para la emisión de certificados de tipo público reconocidos por los dispositivos en el mercado. Amazon Web Services utiliza su propia autoridad certificadora pública, ampliamente reconocida desde la versión service pack 2 de Windows XP.

ACM es un servicio que es capaz de gestionar todo el ciclo de vida de una autoridad certificadora, así como todos los procesos. No solo incluye la capacidad de emisión y gestión de nuevos certificados de tipo X.509 así como la gestión entera del ciclo de vida de estos, destacando la capacidad de automatizar la renovación o invalidación de los mismos. Una de las características más interesantes es su capacidad de integrarse con servicios nativos de AWS que utilizan SSL/TLS para encriptar el tráfico en transito (balanceadores de carga, gestores de APIs entre otros) para automatizar el despliegue de los certificados y su posterior renovación. ACM también soporta la importación de certificados de tipo público a la herramienta, actuando de esta manera como un almacén seguro para claves privadas.

ACM ofrece los mismos servicios para la gestión de una autoridad certificadora de tipo privado. En este caso podremos importar o crear una autoridad certificadora subordinada si así es nuestro deseo, y a partir de este momento gestionar todo el proceso del ciclo de vida de los certificados.

2.3.2. Microsoft Azure

Al igual que los otros proveedores de nube pública, Microsoft Azure dispone de una oferta de servicios de criptografía que podemos dividir de la misma manera que en AWS en servicios de Criptografía y servicios de PKI. En este caso Microsoft Azure ofrece dos servicios principales, el primero es Azure Dedicated HSM, y el seguro se denomina Microsoft Azure Key Vault el cual ofrece servicios de gestor de claves y se integra con servicios de Microsoft Azure, y una segunda funcionalidad principal en la que opera como una PKI.

2.3.2.1. Microsoft Azure Dedicated HSM

En esta ocasión lo que Microsoft ofrece es un módulo criptográfico de Thales denominado Luna 7 A790. Este dispositivo hardware es totalmente dedicado para el cliente de manera que no se comparte en ninguna circunstancia, esto le permite ofrecer un nivel de certificación tipo FIPS 140-2 nivel 3 y eIDAS criterios comunes EAL4+. En este caso es un servicio que funciona de la misma manera que si adquiriéramos dispositivos de Thales, y de hecho Microsoft referencia a la documentación del propio fabricante de cara a la configuración y administración de estos módulos hardware. En este caso el servicio de Microsoft Azure Dedicated HSM tiene un propósito muy especifico de ofrecer soporte a necesidades particulares de nuestras aplicaciones fundamentalmente debido a algún tipo de requisito de seguridad o de cumplimiento especifico ya que este servicio no se integra con otros servicios de Azure como son los servicios de base de datos, almacenamiento o incluso el propio servicio de Azure Key Vault.

Microsoft Azure Dedicated HSM es una buena solución para hacer tareas de off-loading para servicios HTTP que usan TLS/SSL, encriptar datos, establecimiento de una PKI, gestión de servicios de DRM o firma de documentos. Sin embargo, otros servicios como pueden ser transacciones financieras no son ofrecidos por parte de este tipo de dispositivos, que como en en resto de proveedores mantienen una aproximación generalista.

Aunque es un servicio gestionado por Microsoft, en este caso solo ofrecen la capacidad de despliegue del elemento sin más opciones, el resto son asumidos por el cliente hasta el punto de que no se ofrecen acuerdos de nivel de servicio y las herramientas que utilicemos (SDKs, APIs, documentación) deben ser gestionadas con el fabricante Thales.

2.3.2.2. Microsoft Azure Key Vault (AKV)

Azure Key Vault o AKV es el servicio de criptografía gestionado ofrecido por Microsoft Azure. Este servicio se integra de manera nativa con los principales servicios de Microsoft Azure que disponen de capacidades para almacenar datos de manera permanente con el objetivo de proporcionar la capacidad para encriptar y desencriptar en tiempo real los datos y protegerlos de esta manera. Así mismo Azure Key Vault permite su funcionamiento como una PKI de manera que podremos emitir certificados X.509 garantizados por una CA pública como DigiCert (en este caso debemos integrarnos con una autoridad certificadora pública externa a Microsoft). Microsoft Azure Key Vault realizará todas las tareas necesarias para una PKI, como la emisión de certificados, renovación, expiración, validación online, o revocación. Así mismo actuará como un servicio de almacenamiento de claves seguro.

AKV utiliza un esquema de gestión de claves similar al ofrecido en AWS por KMS, con una diferencia principal en la que la manera de proteger la clave que encripta los datos (Data Encryption Key) en este caso se protege mediante una clave asimétrica y no simétrica como ocurría en AWS KMS, esta clave puede ser generada por Microsoft o puede ser importada por el cliente y se denomina Key Encryption Keys.

Otra diferencia importante entre AKV y AWS KMS es el mecanismo de rotación de claves. En el caso de AKV todas las rotaciones de claves se hacen por parte del cliente con la generación de un evento cuando el periodo de rotación se acerca y se ejecuta una función para realizar esa rotación. Sin embargo, en AWS como se describió la rotación en el caso de la clave gestionada por el cliente ocurre automáticamente.

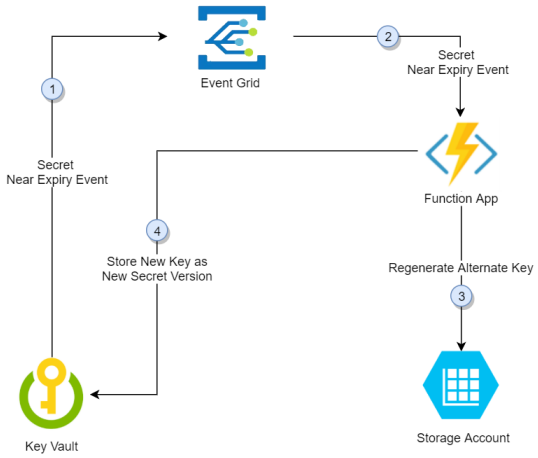


Ilustración 3 - Arquitectura Azure Key Vault

2.3.3. Google Cloud

El caso de Google Cloud es muy similar a la oferta de servicios de otros proveedores de servicios en la nube. Ofrece como servicios principales de criptografía la posibilidad de usar un servicio virtual denominado Google Cloud KMS el cual puede ser complementado por un servicio dedicado para el alojamiento de claves llamado CloudHSM. En el caso de Google hay una diferencia fundamental en estos servicios, y se trata de CloudHSM el cual, aunque al igual que en Microsoft Azure o Amazon Web Services parece que es un servicio de Hardware dedicado single-tenant para el cliente Google no lo especifica y este no es directamente consumible por el cliente, y debe usar como capa intermedia el servicio de Google Cloud KMS para acceder a los servicios. Esto implica que en este caso CloudHSM de Google solo actúa como un almacén de claves seguro sobre hardware dedicado y es útil en los casos en que el cliente debe cumplir con FIPS 140-2 nivel 3.

Google también ofrece un servicio de PKI, pero en este caso solo cubre la situación de una PKI privada y no se soporta la integración con una CA de tipo público.

2.3.3.1. Google Cloud HSM

Google CloudHSM es un módulo criptográfico HSM con un nivel de cumplimiento FIPS 140-2 nivel 3. Este módulo permitirá el alojamiento de claves de encriptación, así como la ejecución de operaciones criptográficas. Google se responsabilizará de toda la operación del equipo HSM incluyendo el despliegue, mantenimiento, actualizaciones, administración, integración con Google KMS, y escalado de la plataforma en base a la demanda. En este caso Google no describe si el elemento HSM es single-tenant, es decir dedicado al cliente o no, por su mecanismo de funcionamiento se puede deducir que lo que en realidad nos entrega Google es una partición de su servicio CloudHSM y no un hardware dedicado.

El uso del producto como se describe se realiza a través del servicio Google KMS usando sus API y simplemente pidiendo un nivel de protección HSM, este comando ilustra el mecanismo:

```
gcloud kms keys create key \
    --keyring key-ring \
    --location location \
    --purpose "encryption" \
    --protection-level "hsm"
```

Ilustración 4 - Comando Google KMS

Google CloudHSM actúa únicamente como un HSM gestionado por el servicio KMS para el almacenamiento de claves en un dispositivo que cumple la norma FIPS 140-2 nivel 3.

2.3.3.2. Google Key Management Service (KMS)

Google KMS es un servicio gestionado al igual que los descritos para Amazon Web Services y Microsoft Azure, que se integra con los servicios nativos de Google y permite dotarles a cualquier escala de servicios de criptografía. Empezando por la generación, almacenamiento, y gestión del ciclo de vida de claves, tanto simétricas como asimétricas. Google KMS también gestiona las operaciones criptográficas necesarias para estos servicios tanto de encriptación como de desencriptación a la escala que sea demandada.

Google KMS ofrece una interesante característica que no encontramos en ninguna del resto de las ofertas de proveedores de nube pública que se denomina “External Key Manager” la que permite integrar con KMS un almacén de claves externo como puede ser un HSM on-premises. Esto dota a los clientes de sencillez a la hora de mantener su capacidad actual de administrar y manejar sus claves en dispositivos HSM mientras utilizan para su consumo un servicio como Google KMS que realiza la integración con los servicios nativos de Google como BigQuery, Google Kubernetes Engine o su servicio de almacenamiento.

Al igual que los otros servicios gestionados de claves del resto de proveedores Google KMS permite la creación de políticas de rotación de claves, importación de claves externas, demorar la destrucción de las claves (por política), gestionar claves simétricas y asimétricas, así como un amplio portfolio de algoritmos de encriptación, firma y atestación.

2.3.4. Comparativa entre servicios de nube pública

2.3.4.1. Sistemas HSM con hardware dedicado

Característica	AWS	Microsoft Azure	Google Cloud
HW dedicado	Si	Si	No
Certificaciones	FIPS 140-2 nivel 3	FIPS 140-2 nivel 3 EIDAS cc EAL4+	FIPS 140-2 nivel 3
Capacidad de clustering	Si	Si, gestionado por el cliente	n/a

Rendimiento	30.800 ops RSA 8.820 ops ECC pmul/sec 8.400Mb/s full-duplex AES-CGM	10.000 ops RSA 20.000 ops ECC 20.000 ops AES-CGM	3.000 QPM para operaciones Asimétricas 30.000 QPM para operaciones simétricas

2.3.4.2. Comparativa funcional HSM en la nube

Soporte	AWS	Microsoft Azure	Google Cloud
Claves	• RSA – 2048-bit to 4096-bit RSA keys, in increments of 256 bits. • AES – 128, 192, and 256-bit AES keys. • ECC key pairs for NIST curves secp256r1 (P-256), secp384r1 (P-384), and secp256k1 (Blockchain).	• RSA – 2048-bit, 3072-bit, 4096-bit RSA keys • AES – 128, and 256-bit AES keys. • ECC key pairs for NIST curves: EC-P256, EC-P256K, EC-P384, EC-521	• RSA – 2048-bit, 3072-bit, 4096-bit RSA keys • AES –256-bit AES keys. • ECC key pairs for NIST curves: EC-P256, EC-P256K, EC-P384, EC-521
Algoritmos	• AES-CBC • AES-ECB • AES-CTR • AES-GCM • AESWrap-ECB • DESede-CBC (TripleDES) • DESede-ECB • RSA-ECB • RSA-AES-Wrap	• AES-CBC • AES-KW • AES-GCM • RSA-OAEP • RSA-OAEP-256 • RSA-PKCS#1v1.5	• AES-GCM • RSA-OAEP • RSA-OAEP-256
Digests	• SHA-1 • SHA-224 • SHA-256 • SHA-384 • SHA-512	• SHA-1 • SHA-2 • SHA-3 • SM2 • SM3 • SM4	• SHA-1 • SHA-256 • SHA-384 • SHA-512
Hash	• HmacSHA1 • HmacSHA224 • HmacSHA256 • HmacSHA384 • HmacSHA512	• HmacSHA1 • HmacSHA2 • HmacSHA3 • HmacSM2 • HmacSM3 • HmacSM4	• HmacSHA256
Modos de firma	• RSA-PSS • RSA-PKCS#1v1.5 • ECDSA with P256 • ECDSA with P384 • ECDSA with P512 • ECDSA with SECP-256k1	• RSA-PS256 • RSA-PS384 • RSA-PS512 • RSA-RS256 • RSA-RS384 • RSA-RS512 • RSA-RSNULL • ECDSA with P256 • ECDSA with P384 • ECDSA with P512 • ECDSA with SECP-256k1	• RSA-PSS • RSA-PKCS#1v1.5 • ECDSA with P256 • ECDSA with P384 • ECDSA with SECP-256k1

3. Infraestructura de Clave Pública

La infraestructura de clave pública es el conjunto de elementos necesarios para permitir la creación de los certificados digitales que permitirán facilitar la transferencia de información entre diferentes sistemas.

Los casos de uso de una Infraestructura de Clave Pública o PKI por si siglas en ingles son variados y entre ellos cabe destacar los siguientes:

- Certificados SSL/TLS que permiten hacer segura la navegación o comunicación entre sistemas
- Firma digital para aplicaciones
- Autenticación para sistemas como VPN, login de usuarios, o autenticación sistema a sistema (máquina a máquina)
- Encriptación de datos como correos electrónicos o documentos
- Como complemento en procesos de autenticación en sistemas WiFi

Como se puede ver una PKI puede ser un elemento fundamental en la identificación de una persona o una máquina, así como ser un elemento crítico para mantener la privacidad de los datos mediante su encriptación en transito. Debido a esto, se introduce un nuevo elemento, disponer de mecanismos de confianza en estos certificados, de manera que podemos garantizar que quien se autentica es quien dice ser, o que con quien estamos hablando igualmente es el sistema con el que queremos hablar.

Las PKI introducen un concepto denominado autoridad certificadora, la cual se configura como un mecanismo, generalmente proporcionado por un tercero, dentro de los servicios de PKI que van a garantizar que ese certificado que estamos usando para cualquiera de los propósitos es autentico y que podemos confiar en el. Por tanto, las autoridades certificadoras son autoridades que equivaldrían a lo que podría ser un notario on-line (sin estas propiedades legales en muchos casos y si en otros) que garantiza que ese certificado con el que estamos hablando es quien dice ser.

Tenemos dos tipos de Autoridades Certificadoras, el primer tipo es lo que denominamos autoridades públicas, estas autoridades están reconocidas ampliamente en el mercado y forman parte de la mayoría de los sistemas. Además, estas autoridades certificadoras públicas, pueden en algunos casos proporcionar validez legal a los casos de uso para los que emiten certificados, como puede ser la autoridad certificadora de la Policía Nacional en España, o la Fábrica Nacional de Moneda y Timbre. En segundo lugar, podemos encontrar lo que se denominan autoridades certificadoras privadas, que generalmente son las que gestionan los certificados dentro de una empresa y que es capaz de realizar la emisión y validación de estos certificados de forma centralizada para los sistemas de la compañía e incluso si así lo acordamos con terceros. Este tipo de autoridades certificadoras privadas son de confianza dentro de nuestro ámbito de actuación que será generalmente nuestra empresa o terceros con los que interactuemos.

Por tanto, si revisamos nuevamente que es una PKI podemos decir que es el conjunto de hardware, software, políticas, procedimientos y roles necesarios para poder realizar la emisión, almacenamiento, administración y gestión del ciclo de vida de los certificados digitales que prestarán estos servicios descritos. Así mismo, realizarán otros servicios como es la renovación de estos certificados, validación de los mismos on-line, o revocación de los mismos en caso que por algún motivo se hayan visto comprometidos en su seguridad.

Como se puede deducir de la introducción realizada, la generación, y custodia de las claves o certificados, en particular de las claves privadas de los certificados se convierte en una actividad altamente crítica ya que, de verse comprometida, cualquiera en posesión de esta podría suplantar la identidad de una persona o sistema, con las consecuencias de seguridad e incluso legales que esto podría tener. Por este motivo, el uso de equipamiento especializado es de vital importancia, equipamiento que nos permita almacenar estos certificados y sus claves de manera segura y que nos ayude en la gestión del ciclo de vida de los mismos por ejemplo para su renovación o llegado el caso revocación.

3.1. Análisis de servicios de nube pública para una PKI

En el anterior capitulo hemos visto las diferentes capacidades criptográficas que ofrecen los proveedores de nube pública pudiendo diferenciar estos en tres capas diferentes:

- Servicios Criptográficos no nativos: los cuales en general están compuestos por hardware dedicado al cliente en forma de módulo criptográfico HSM.
- Servicios Criptográficos nativos: estos en general están soportados por un hardware criptográfico pero que no es exclusivo para el cliente altamente escalable y que se integra con los servicios del proveedor de nube de forma nativa, pero que por el contrario no ofrece demasiados servicios para la integración con terceras partes.
- Servicios de PKI: como es el caso de Amazon Web Services con su servicio Amazon Certificate Manager o Microsoft Azure con su servicio Key Vault que como una de las funcionalidades ofrece esta capacidad de PKI, y por último Google Cloud con su servicio de Certificate Authority Service.

Tras la revisión de las funcionalidades de cada uno de ellos, queda claro que los servicios criptográficos nativos no son una buena opción por el límite que tienen en su gestión y por que no están diseñados para esta función. En general estos servicios están orientados a la creación y gestión de claves de tipo simétricas para la encriptación de datos en reposo, que posteriormente serán encriptadas por una clave simétrica (AWS) o por una asimétrica (Azure) para su protección. La emisión de certificados, con las propiedades necesarias para esto no es una funcionalidad para la que podamos utilizar ninguno de los servicios gestionados nativos de los proveedores de nube pública.

La opción sería utilizar los servicios de PKI nativos de estos proveedores, para el objetivo definido en este trabajo podríamos utilizar los servicios de PKI de clave privada lo que nos permitiría disponer de nuestra autoridad certificadora completa desde la CA raíz, así como las subordinadas y finalmente disponer de la capacidad para la emisión de los certificados.

Uno de los objetivos del trabajo es entender la viabilidad y "transportabilidad" de las claves, así como de la PKI entre nubes o incluso entre sistemas on-premise, por lo que las autoridades certificadoras como ACM de Amazon o Key Vault de Azure suponen un limite a esto ya que ambas solo se pueden consumir con APIs propietarias y no estándares de mercado como PKCS#11 o NCG de Microsoft.

Por lo tanto, nos queda una aproximación basada en equipos HSM los cuales en general ofrecen una capacidad de integración más estándar, y de cara a los servicios nativos en la nube en general solo actúan como almacenes de claves y en algún caso como servicio de off-loading de operaciones criptográficas. De los tres elementos analizados, Google CloudHSM queda fuera de la foto ya que solo puede ser consumido a través de Google KMS lo que limita su usabilidad como parte de una PKI. En el caso de Microsoft el servicio ofrecido se podría utilizar sin mayor problema ya que es un dispositivo bastante estándar en la industria como es el Luna 7 de la empresa Thales.

Como lo que buscamos es determinar la viabilidad de levantar una PKI con las características de la nube la mejor opción es utilizar el servicio de CloudHSM de AWS que nos permite obtener todas las funcionalidades de un HSM, pero perfectamente integrado con el ecosistema de AWS y con amplias capacidades de integración vías protocolos estándar como PKCS#11.

3.2. Requisitos técnicos para una PKI en la nube

Una PKI tiene diferentes aspectos a considerar, por ejemplo, las ceremonias de claves son vitales en la operación de cualquier PKI, pero debido a la naturaleza técnica de este trabajo no los vamos a evaluar. Sin embargo, nos centraremos en el resto de elementos en los que a continuación vamos a definir los requisitos técnicos para ser capaces de desplegar de manera automatizada una PKI en la nube.

Una PKI en general está compuesta por diferentes jerarquías. Las diferentes jerarquías representan diferentes niveles de autoridad dentro la PKI. En las PKI de carácter público, la autoridad de certificación raíz, en general es una autoridad pública ampliamente reconocida que será la responsable bien de certificar toda la información en los certificados, o de delegar esta capacidad en un tercero en el que confía y con un ámbito determinado (por ejemplo, una empresa dentro de su dominio de empresa).

La autoridad certificadora subordinada se apoya en esta autoridad certificadora raíz que le ha delegado esa capacidad de emitir certificados y que estos van a ser respaldados mediante una cadena de certificados por la autoridad certificadora raíz. Esto sería lo que llamamos una PKI de dos capas. Y la arquitectura que representaría esta arquitectura se muestra en la siguiente imagen:

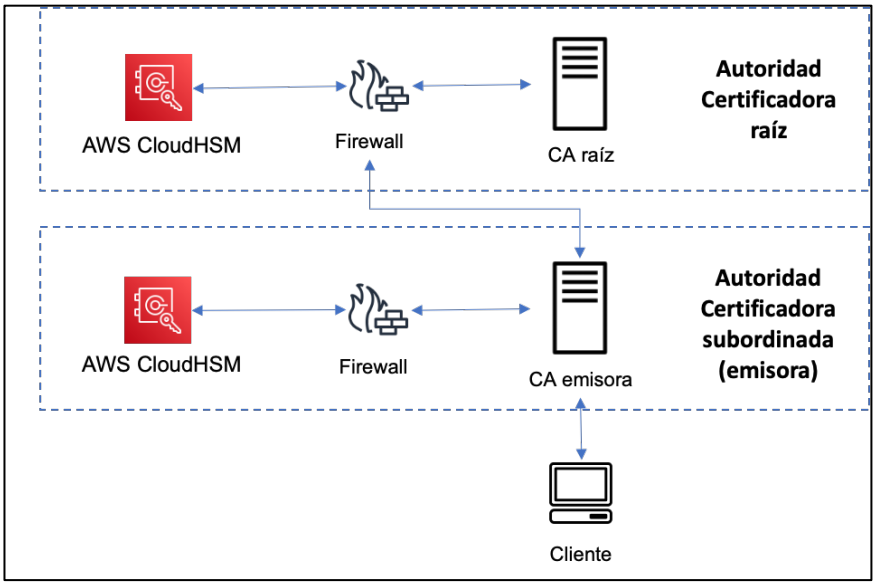


Ilustración 5 - Arquitectura CAs con CloudHSM

Esta arquitectura nos permite utilizarla para diferentes casos de uso y de manera modular. Si nuestra autoridad certificadora es privada, la autoridad certificadora raíz, será el elemento principal de nuestra PKI privada, que validará todos los certificados y será parte de la cadena de confianza que nuestros dispositivos deberán utilizar. Si lo que buscáramos es desplegar una autoridad certificadora, pero de tipo público la capa de la CA raíz la deberíamos sustituir por los servicios de una CA reconocida de tipo público que nos daría

su certificado que deberemos almacenar de manera segura en el HSM de la autoridad certificadora subordinada.

Pero una autoridad certificadora no solo está compuesta de estos elementos HSM que como hemos descrito son claves para la generación de certificados, almacenamiento seguro, gestión del ciclo de vida y revocación de los mismos. Aunque importantes una PKI contiene adicionalmente los siguientes elementos.

3.2.1. Autoridad de validación

La autoridad de validación es el servicio que nos permitirá confirmar que los certificados siguen siendo aceptables en el tiempo. Para esto se utilizan dos técnicas diferentes, en primer lugar, el uso de "Certificate Revocation Lists" o CRL que simplemente está compuesto por una lista con los números de serie de los certificados que no son válidos por cualquier motivo (revocados, invalidados). En segundo lugar, disponemos de un protocolo de validación denominado Online Certificate Status Protocol u OCSP, este protocolo funciona sobre protocolo HTTP de manera online, y es una alternativa a CRL que permite casi en tiempo real validar que un certificado no ha sido revocado por cualquier razón.

OCSP está en general dirigido a la validación de manera automática de estos certificados y lo usan de manera amplia los navegadores de clientes finales, o los servidores de aplicaciones. Debido a que se trata de un protocolo on-line y que se busca un enfoque de valido o no valido la información que se obtiene es la mínima necesaria para validar el certificado. Sin embargo, CRL proporciona una cantidad de información mucho más rica en cuanto por ejemplo los motivos por los que el certificado ha sido invalidado.

3.2.2. Autoridad de registro

El proceso de emisión de certificados siempre lo inicia el cliente que quiere obtener un certificado, ya puede ser un equipo informático de un usuario final que quiere conectarse a una red Wifi, o autenticarse con una página web, o puede ser un sistema que desea registrarse y poder hablar de manera segura y autenticado con otro sistema. Para esto, el sistema necesita emitir lo que se llama una petición de firma del certificado "CSR" y enviarlo a la autoridad de registro. La autoridad de registro será la responsable de validar esa petición, en especial los datos de identificación contenidos en la solicitud del certificado y si esto es valido entonces firmar ese certificado como válido siendo a partir de este momento reconocido por cualquier sistema que confíe en esta CA como un certificado válido al tener la validación de la autoridad certificadora.

3.3. Componentes de la PKI

Tras revisar los diferentes elementos que componen una PKI vamos a definir que elementos son los que utilizaremos en forma de hardware y software para la construcción de nuestra PKI. Es cierto que faltarían dos elementos adicionales, documentos y procedimientos.

Los documentos determinarán que tipo de certificados y para que propósito podemos utilizar estos, por ejemplo, autenticar en una red wifi, contra un servidor web usando TLS, firmar digitalmente un correo electrónico, entre otros. Una vez que definamos la solución software que vamos a utilizar para manejar nuestra PKI definiremos que tipo de documentos aceptaremos.

Los procedimientos indican como se realizan diferentes procesos dentro del marco de la PKI, por ejemplo, como se validan los datos incluidos en el certificado por el peticionario,

como se realiza la firma del certificado, y así cualquier otra actividad. Esto es vital ya que es el mecanismo que tienen las autoridades de certificación de garantizar que sus procesos están bajos las más estrictas garantías de seguridad y sus clientes pueden confirmar en ellos y en la seguridad de estos. Para el presente trabajo, dado su carácter experimental esta parte está fuera del alcance del mismo.

3.3.1. Hardware seguro

Tras la revisión que hemos realizado de la oferta de servicios criptográficos de los diferentes proveedores de nube, hemos seleccionado a AWS CloudHSM como el mejor elemento para construir una PKI en la nube. El motivo es la flexibilidad que ofrece con su integración del SDK con PKCS#11 el cual es un standard ampliamente aceptado en la industria y nos ofrece una gran compatibilidad para el objetivo del presente trabajo.

3.3.2. Software de la PKI

Siguiendo la naturaleza de este trabajo hemos optado por utilizar un software de tipo Open Source. Hemos elegido el software de EJBCA.org ya que ofrece un amplio ecosistema de soporte y cubre todas las necesidades que hemos definido para nuestra PKI: gestión del ciclo de vida del software, autoridad de validación y autoridad de registro. Este software se encuentra disponible bajo licencia GNU Lesser Public License 2.1 que lo hacen perfecto para este trabajo.

Adicionalmente EJBCA nos permite en nuestra PKI emitir un amplio número de certificados de diferente propósito lo que nos proporcionaría una PKI rica en funcionalidades:

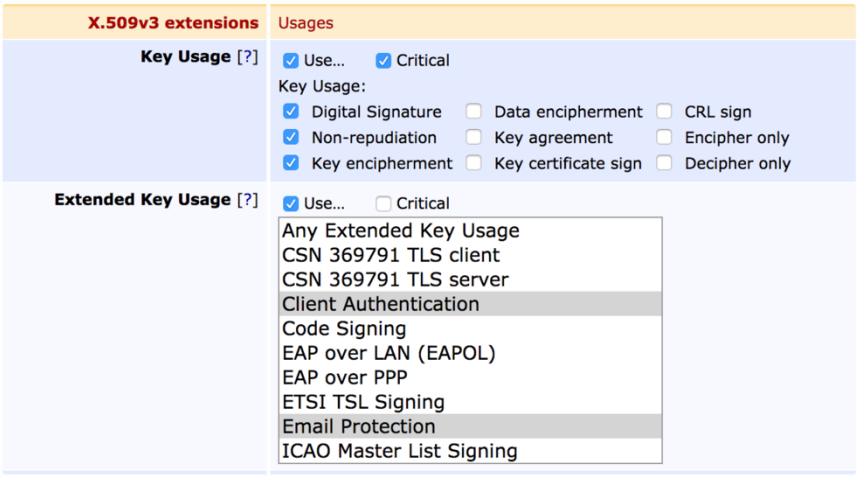


Ilustración 6 - EJBCA tipos de Certificado

3.3.3. Sistema de automatización de emisión de certificados

Uno de los objetivos del presente trabajo es además de ver la interoperabilidad de la arquitectura definida en la nube con sistemas fuera de esta o entre otras nubes, determinar la capacidad de automatización de la emisión de certificados por parte de la PKI para sistemas que se comunican máquina a máquina. Para ellos vamos a utilizar como prueba de concepto una arquitectura de microservicios en malla que requerirá de esta capacidad para emitir certificados casi en tiempo real por parte de la PKI. Para ello vamos a utilizar el protocolo ACME (Automated Certificate Management Environment) que será

responsable de por un lado solicitar a la PKI la emisión de nuevos certificados, y por otro lado de proporcionárselos al sistema que los está solicitando de manera automática.

Por este motivo es importante que el software de PKI que utilicemos tenga soporte para este protocolo que automatiza la emisión de certificados, y así mismo el origen que solicite los certificados también debe ser compatible con el modo de solicitud ACME. Como el diseño de la arquitectura está basado en microservicios en forma de contenedores, hemos validado que el servicio nativo cert-manager.io soporta la solicitud de certificados usando ACME.

3.3.4. Arquitectura propuesta

Con estos requisitos y elementos ya definidos, la arquitectura propuesta es la siguiente:

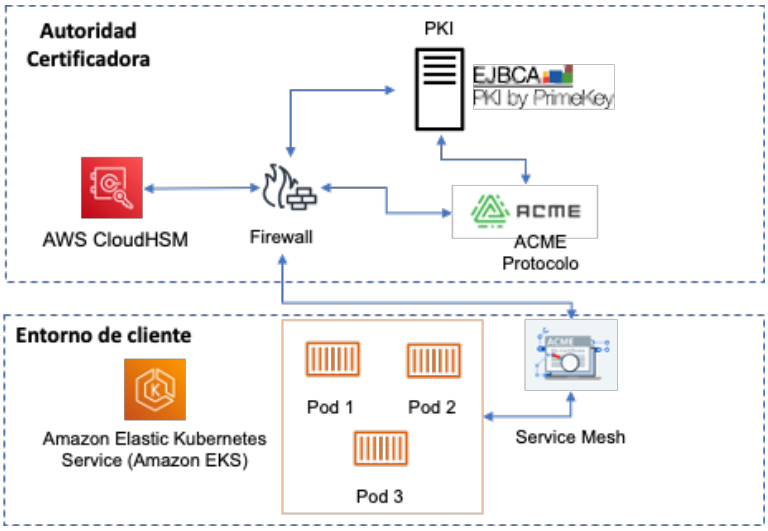


Ilustración 7 - Arquitectura propuesta alto nivel

Como se puede observar en la arquitectura el cuadro superior representa el servicio completo de la PKI incluyendo la capacidad de consumirla vía protocolo ACME para la automatización de la emisión de certificados. Este consumo automático será orquestado por un sistema de malla de aplicaciones.

En la parte superior de la autoridad certificadora, vemos como se utiliza el servicio de CloudHSM como elemento de seguridad para almacenar los elementos críticos como claves y certificados. EJBCA como software de PKI que es consumible vía interface WEB o API, y con este último ofrece entre otros el protocolo ACME para la emisión automatizada de certificados.

4. Diseño e implementación de la solución

4.1. Descripción del modelo a desplegar

El objetivo de este trabajo es validar la posibilidad de desplegar una PKI en la nube pública con todas las medidas de seguridad necesaria y luego mostrar su consumo en un caso de uso como es el de una infraestructura de contenedores y aprovechar una de sus capacidades en forma de Service Mesh para gestionar la emisión automatizada de certificados para los contenedores.

En este diseño vamos a aprovechar algunas de las ventajas que AWS nos ofrece en forma de aislamiento y para ello desplegaremos nuestra PKI apoyados en los servicios descritos durante este documento, pero de manera aislada en una VPC (VPC de trabajo PKI) propietaria, desde la que expondremos estos servicios como si un proveedor se servicios se tratara a las infraestructuras que deseen consumirlos (nuestro caso de uso, por ejemplo).

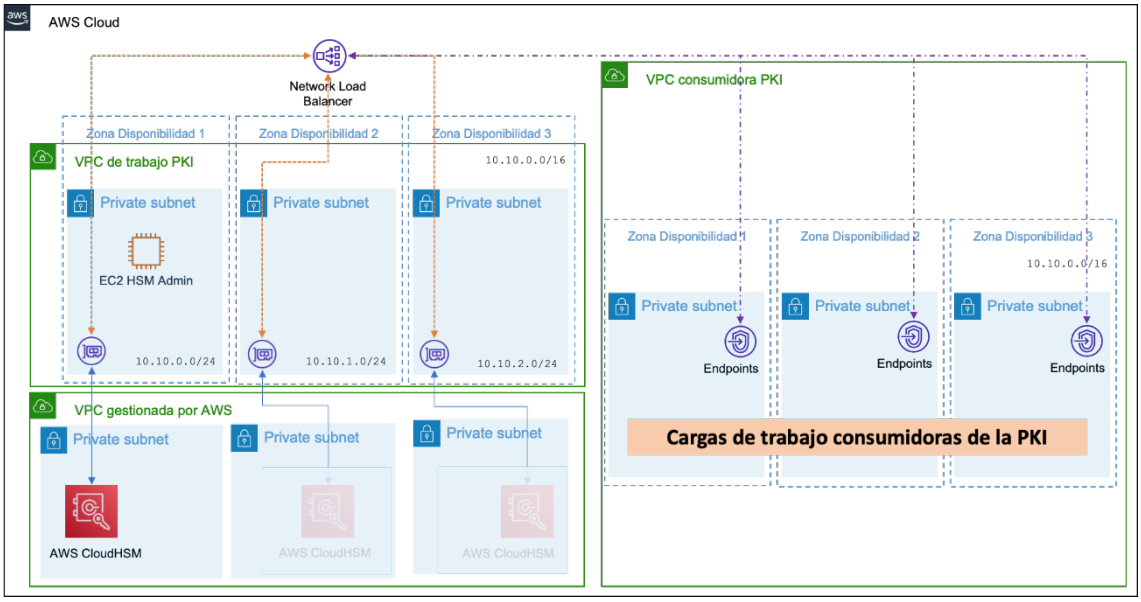


Ilustración 8 - Arquitectura base solución en AWS

El diseño que se ha realizado como se ve tiene tres áreas separadas que se denominan VPC (Virtual Private Cloud). Este elemento es un contenedor lógico diseñado por AWS para aislar las diferentes cargas de trabajo que no tienen comunicación entre ellas salvo que los usuarios así las construyan.

La primera VPC denominada “VPC gestionada por AWS” es donde se desplegarán los elementos HSM que son gestionados por AWS. Esta VPC no es accesible ni configurable por el usuario, solo AWS tiene acceso a ella y es parte de lo descrito como modelo de responsabilidad compartida que asume AWS. En esta VPC se despliega toda la configuración del clúster de HSM, y dentro de este clúster puedes desplegar hasta un máximo de 28 dispositivos CloudHSM.

Otro elemento importante es entender que los elementos HSM, aunque formen parte del mismo clúster deben ser desplegados en lo que AWS denomina Zonas de disponibilidad, a partir de aquí lo que hará AWS es conectar ese HSM gestionado a la zona de disponibilidad equivalente en nuestra VPC mostrando en ella simplemente la interface de

red que utilizaremos para acceder al HSM. Para el diseño y demostración solo utilizaremos una de ellas por razones económicas, y ya que el objetivo es la demostración funcional de la solución.

La segunda VPC, denominada “VPC de trabajo PKI” es donde desplegaremos todos nuestros servicios de PKI. En primer lugar, protegeremos los puntos de acceso al servicio HSM que como se ha explicado se presentan en forma de interfaces de red, pero admiten la configuración de un elemento denominado grupos de seguridad que actúan como un Firewall.

La comunicación de los servicios se realiza en los puertos 2223 a 2225 con protocolo TCP. Esta regla se utiliza tanto para el cliente de administración de CloudHSM, como para la comunicación entre si de los diferentes miembros del clúster. Así que en este grupo de seguridad deberemos permitir la comunicación con origen otros nodos del clúster, o la máquina virtual donde tenemos las herramientas de administración (instancia EC2 en el diagrama).

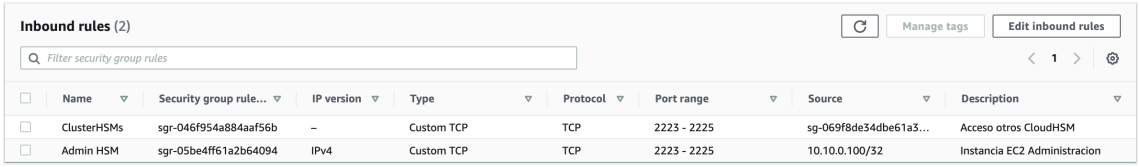


Ilustración 9 - Ejemplo Grupo de Seguridad para HSM

Como se puede observar en el diagrama, este es el grupo de seguridad que asignaremos a los puntos de conexión que CloudHSM desplegará en nuestra VPC. La primera regla hace referencia a ella misma, por lo que cualquier HSM que tenga asignada esta regla podrá hablar con cualquier otro. La segunda regla limita el acceso a los HSM únicamente a la instancia EC2 de administración que hemos definido (máquina virtual).

La máquina virtual desplegada en la VPC de la PKI será le herramienta de administración principal de los elementos HSM. Esta máquina no es necesaria para la producción o prestación de los servicios de PKI, pero será imprescindible para varias tareas como la inicialización del clúster y generación de la CA raíz, establecimiento y configuración del clúster, administración de los usuarios (administrador, operador, etcétera) y gestión y acceso a las operaciones criptográficas de generación, eliminación o uso de las diferentes claves soportadas por el servicio de CloudHSM.

Para su correcto funcionamiento en este clúster se instalarán las librerías necesarias que otorgan la compatibilidad con el mismo, así como las herramientas de administración necesarias.

Una vez descrita la conectividad entre la VPC en la que prestaremos servicios de PKI, ahora veremos como estos accesos son ofrecidos a terceros sin entrar aun en detalles de los componentes como son las aplicaciones que gestionarán la creación y emisión de certificados, así como las operaciones criptográficas necesarias, en concreto EJBCA. Para ofrecer estos servicios desde la VPC de la PKI utilizaremos un mecanismo que AWS denomina como AWS PrivateLink y que permite en un solo sentido exponer servicios (solo son consumibles desde el exterior, desde la VPC de la PKI no se puede acceder proactivamente a las otras VPC). El mecanismo de funcionamiento implica desplegar un balanceador de carga de red (funciona en las capas OSI nivel 3 y 4) que utilizará como destinos los puntos de conexión desplegados por los HSM o la PKI, y expondrá su parte frontal mediante el mecanismo de PrivateLink en la VPC de nuestro cliente, que no hace

otra cosa que desplegar una tarjeta de red en la VPC y las diferentes zonas de disponibilidad de esta que permiten el acceso al servicio.

4.2. Despliegue de EJBCA

EJBCA es un software de código abierto que ofrece todas las funcionalidades necesarias para establecer una PKI. Este software se ofrece en diferentes versiones (versión comunidad, y versión Enterprise) las cuales se diferencian fundamentalmente en las capacidades de integración con terceros, las cuales solo son ofrecidas en la versión empresarial. A continuación, se puede ver la tabla de comparación de las diferentes características entre las dos versiones:

Característica	Enterprise	Comunidad
Licencia	Open Source LGPL v2.1 o posterior	Open Source LGPL v2.1 o posterior
Características de PKI	Completa, incluye todos los protocolos	Completa, incluye todos los protocolos
Recomendado para…	EJBCA Enterprise se recomienda para corporaciones, gobiernos y otras organizaciones que buscan una solución de escala empresarial, lista para producción, certificada, de código abierto para la PKI sin licencias iniciales.	EJBCA edición comunidad está recomendada para desarrolladores y usuarios técnicos de PKI en entornos que no sean de misión crítica. Esta versión no tiene soporte y solo se pretende que sea utilizada por personas que tengan tiempo para resolver sus problemas ellos mismos.
Perfecto para…	EJBCA es perfecto para pequeñas a grandes despliegues de PKI desde 1.000 a más de 100 millones de certificados emitidos.	EJBCA solo es adecuado para despliegues pequeños y a gran escala de PKI de 1.000 a más de 100 millones de certificados emitidos.
Certificaciones de seguridad	EJBCA Enterprise ha sido certificado bajo el Common Criteria EAL 4+ (CIMC Protection Profile) y CWA 14167-1 (en instalación de cliente)	Ninguno
Soporte comercial	PrimeKey proporciona soporte comercial con acuerdos de nivel de servicio (SLA) para la resolución de incidentes, parches y soluciones.	No se provee soporte, existe soporte de la comunidad a través de foros y listas de correo.
Auditoría con integridad protegida	EJBCA Enterprise dispone de la certificación Common Criteria en el mecanismo de auditoría utilizando HMAC o firmas digitales para la protección de la integridad de los logs	No
Protección de la integridad de la Base de Datos	EJBCA Enterprise dispone de protección para la integridad de la base de datos certificada bajo el Common Criteria anti DBAs maliciosos	No
Test de penetración	EJBCA Enterprise ha sido probado con test de penetración como parte de la certificación de Common Criteria así como por terceras partes	No

Separación de roles	Separación de roles completa incluyendo la interface de línea de comandos	Separación de roles para usuarios con acceso remoto
Proceso de remediación de bugs	PrimeKey dispone de un proceso de seguimiento de bugs de seguridad y otro tipo de incidentes evaluado bajo el marco de Common Criteria.	La comunidad EJBCA sigue mediante un mecanismo de desarrollo abierto y seguimiento de incidentes si tiempos de respuesta garantizados.
Precio de licencia / Suscripción	No hay coste por la licencia – Se proporciona como parte de una suscripción anual que corresponde a un nivel de soporte comercial.	No hay coste de licencia de software – Gratuita la descarga y su uso.
Características adicionales	Hot fixes de emergencia, alertas de seguridad, asesoría de buenas prácticas, seguimiento de incidentes privados, guías adicionales y herramientas.	PKI más completa y con más características, con el mejor rendimiento comparado con otras soluciones open source y PKIs comerciales.

EJBCA será el software de PKI que utilizaremos como proveedor de certificados de manera automatizada. Este software entre otras cosas ofrece servicios y soporte nativo mediante conectores automáticos hacía el servicio de AWS CloudHSM.

PKCS#11 es uno de los estándares criptográficos definidos por la RSA, en particular el 11 es el que corresponde a un conjunto de llamadas API comunes que los fabricantes de hardware HSM ponen a disposición de los usuarios para abstraer las particularidades de cada dispositivo y unificar las API de acceso a las diferentes operaciones criptográficas de estos dispositivos. Este mecanismo es el que EJBCA utilizará para comunicarse con los módulos HSM en general, y con AWS CloudHSM en particular.

Uno de los aspectos que deberemos valorar en la implementación final será que parte de las operaciones criptográficas son ejecutadas en el dispositivo hardware, pero sobre todo que parte de los certificados generados (en especial las claves privadas que contienen el material sensible) son gestionadas por el hardware físico.

El despliegue de la solución de EJBCA quedaría de la siguiente manera:

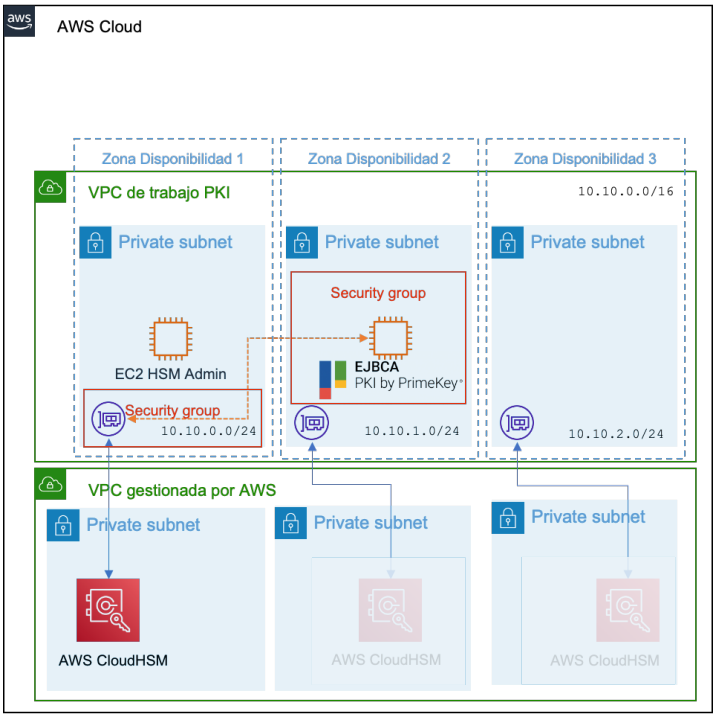


Ilustración 10 - Arquitectura EJBCA en AWS con CloudHSM

Con este despliegue, tendríamos una PKI completamente operativa disponible en AWS en las VPC que como se describen en el anterior punto no serían accesibles directamente por las cargas de trabajo, y que expondrían sus servicios (por ejemplo, acceso a la administración de la PKI) a través de un AWS Privatelink.

4.2.1. Instalación de EJBCA

Se ha optado por la instalación de la versión "Community", aunque algunas funcionalidades solo las obtendremos desde la versión Enterprise, ya que el objeto de este trabajo es mostrar la interoperabilidad en diferentes entornos y no validar que alguna de las soluciones ofrecidas con integración nativa funciona correctamente.

En primer lugar, hemos desplegado como se describe en la parte de la infraestructura de nube las diferentes máquinas virtuales. En la máquina virtual destinada a alojar el software de EJBCA hemos descargado tanto el paquete EJBCA en su versión "7.4.3.2 comunidad" así como un servidor de aplicaciones igualmente OpenSource. Hemos elegido WildFly versión 18.0.0 final, ya que según la documentación de EJBCA es la versión recomendada para esta versión de la PKI.

La instalación del software EJBCA aunque muy bien descrita por el fabricante no es un proceso sencillo ya que no todos los aspectos están bien detallados, el funcionamiento está documentado para que funcione en la primera ejecución, si no es así el diagnostico de problemas es realmente complejo y con poco soporte en la edición comunidad. Aquí vamos a destacar los aspectos claves de la instalación del software de manera que el servidor de aplicaciones WildFly y el software EJBCA quedan alineados para su correcto funcionamiento.

Descargaremos siguiendo las URL disponibles en la documentación los dos paquetes de software lo que descomprimiremos en la carpeta /opt por sencillez de gestión, y sobre todo de seguridad, evitando utilizar directorios personales tipo /home.

Una vez realizado, crearemos los enlaces simbólicos para facilitar el funcionamiento y la actualización de futuras versiones del software, y estaremos listos para comenzar la configuración tanto de EJBCA el propio software como del servidor de aplicaciones.

Los aspectos más críticos en la configuración de ambos, es que las contraseñas de las diferentes keystores estén alineadas entre las que genera EJBCA y las que luego necesita WildFly abrir para el funcionamiento del servidor de aplicaciones. Para ello hay dos contraseñas que van a ser vitales y vamos a tener que definir y mantener: la contraseña de keystore.jks y la contraseña de truststore.jks.

EJBCA como parte de su proceso de instalación creará estos dos keystores, el primero de ellos (keystore.jks) será el responsable de mantener los certificados privados (mientras no tengamos configurado el HSM) de los diferentes usuarios, y el fichero truststore.jks será el que mantenga las entidades CA de confianza en las que nuestro servidor de aplicaciones WildFly se apoyará y confiará que en nuestro caso serán las emitidas y gestionadas en el futuro por EJBCA.

Para la configuración de WildFly en todos los casos hemos utilizado su propia herramienta de configuración llamada Elytron.

Para ello es clave el apartado de configuración de wildFly los siguientes comandos:

```
/opt/wildfly/bin/jboss-cli.sh --connect
'/subsystem=elytron/credential-store=defaultCS:add-
alias(alias=httpsKeystorePassword, secret-value="XXXXXX")'
/opt/wildfly/bin/jboss-cli.sh --connect
'/subsystem=elytron/credential-store=defaultCS:add-
alias(alias=httpsTruststorePassword, secret-value="YYYYYY")'
/opt/wildfly/bin/jboss-cli.sh --connect '/subsystem=elytron/key-
store=httpsKS:add(path="keystore/keystore.jks",relative-
to=jboss.server.config.dir,credential-reference={store=defaultCS,
alias=httpsKeystorePassword},type=JKS)'
/opt/wildfly/bin/jboss-cli.sh --connect '/subsystem=elytron/key-
store=httpsTS:add(path="keystore/truststore.jks",relative-
to=jboss.server.config.dir,credential-reference={store=defaultCS,
alias=httpsTruststorePassword},type=JKS)'
/opt/wildfly/bin/jboss-cli.sh --connect '/subsystem=elytron/key-
manager=httpsKM:add(key-
store=httpsKS,algorithm="SunX509",credential-
reference={store=defaultCS, alias=httpsKeystorePassword})'
/opt/wildfly/bin/jboss-cli.sh --connect '/subsystem=elytron/trust-
manager=httpsTM:add(key-store=httpsTS)'
/opt/wildfly/bin/jboss-cli.sh --connect '/subsystem=elytron/server-
ssl-context=httpspub:add(key-
manager=httpsKM,protocols=["TLSv1.2"],use-cipher-suites-
order=false,cipher-suite-
filter="TLS_DHE_RSA_WITH_AES_128_GCM_SHA256,TLS_ECDHE_RSA_WITH_AES_1
28_GCM_SHA256")'
/opt/wildfly/bin/jboss-cli.sh  --connect  '/subsystem=elytron/server-
ssl-context=httpspriv:add(key-
manager=httpsKM,protocols=["TLSv1.2"],use-cipher-suites-
order=false,cipher-suite-
filter="TLS_DHE_RSA_WITH_AES_128_GCM_SHA256,TLS_ECDHE_RSA_WITH_AES_1
28_GCM_SHA256",trust-manager=httpsTM,need-client-auth=true)'
```

Como se aprecia en los dos primeros comandos, ahí definimos dos secretos que corresponden a la clave del keystore y a la clave del truststore. Estas dos claves es crítico que coincidan con las que definiremos en la configuración de EJBCA ya que cuando este cree esos dos almacenes seguros si no tienen la misma contraseña WildFly no será capaz de leerlos y por tanto utilizarlos. El resto de elementos se encargarán de la configuración de los diferentes accesos que una vez funcionando EJBCA expondrá sus servicios.

Una vez lista la configuración de WildFly procederemos a personalizar la configuración de EJBCA. Esta configuración inicialmente es sencilla y podremos hacerla en base a los archivos de configuración de ejemplo que este nos entrega, o bien utilizando el wizard del que dispone para rellenar la información. En este proyecto hemos copiado dentro del directorio conf/ de ejbca los ficheros de tipo `.sample` que le acompañan a ficheros `.properties` para que sean leídos por el proceso de compilación, configuración e instalación, en concreto hemos modificado cuatro de estos ficheros para nuestra prueba:

- `cesecore.properties`
- `database.properties`
- `ejbca.properties`
- `web.properties`

Hay tres aspectos fundamentales que complementar como paso previo a la instalación.

Fichero cesecore.properties

En este archivo es donde definiremos como mínimo los siguientes aspectos relevantes para el funcionamiento de la herramienta:

- password.encryption.key: esta contraseña segura será la responsable de almacenar encriptadas todas las contraseñas con esta clave.
- ca.keystorepass: aquí determinaremos la clave por defecto de la keystore para la CA, esta es una de las claves que debe estar alineada con las configuradas en Wildfly.
- ca.toolatexpiredate: Definiremos la fecha máxima (o más lejana) en la que podrá expirar el certificado raíz de la CA

El resto de los elementos hasta la configuración si es necesario de un dispositivo hardware HSM no es necesario modificarlos.

Fichero ejbca.properties

Aquí definiremos al menos los siguientes parámetros:

- appserver.home: Ubicación de nuestro servidor de aplicaciones WildFly y donde EJBCA desplegará sus archivos y configuraciones, así como las keystore.
- ejbca.productionmode: definiremos si el sistema está funcionando en producción o es un modelo de pruebas
- ca.cmskeystorepass: Aquí definiremos la contraseña para la keystore del CMS

El resto de los parámetros no es necesario personalizarlo si no lo deseamos.

Fichero install.properties

En este fichero definiremos los datos necesarios para la primera instalación como por ejemplo los campos del certificado raíz de la CA. Estos campos los encontraremos al inicio del archivo, campos como ca.name o ca.dn.

- ca.token: Aquí definimos si es de tipo soft si queremos utilizar la keystore de java para el almacenamiento de certificados o podemos hacerlo de tipo token.PKCS11 para integrar por ejemplo un HSM.

- ca.tokenpassword: Aquí definiremos la contraseña de acceso al sistema HSM
- ca.keyspec: definimos la longitud por defecto de las claves RSA
- ca.keytype: definimos el tipo de clave para la CA de gestión en mi caso he elegido RSA
- ca.signaturealgorithm: definimos el tipo de algoritmo de firma que vamos a utilizar, en mi caso he elegido SHA256 con RSA.
- ca.validity: definiremos la vida útil del certificado de la CA de gestión, en mi caso he elegido 10 años (3650 días)
- ca.policy: definimos el tipo de policy que le aplicamos al certificado de la CA de gestión.

Fichero web.properties

En este archivo definiremos las configuraciones de acceso a la interface web del sistema EJBCA que se integrarán con la configuración previa que hemos hecho en WildFly.

- java.trustpassword: Aquí definimos la contraseña de la truststore que deberá coincidir con la definida en WildFly.
- superadmin.cn: Aquí definiremos el nombre (CN) del súper administrador de la plataforma que registraremos posteriormente en el certificado de cliente para autenticarnos vía mTLS con la misma.
- superadmin.password: la utilizaremos para proteger la keystore con el certificado para el súper administrador y su autenticación mTLS.
- superadmin.batch: aquí definiremos si queremos que el sistema genere el certificado para el SuperAdmin o no.
- httpserver.password: aquí definiremos la clave con la que protegeremos la keystore SSL que luego utilizará WildFly. Es imprescindible que esta coincida con la definida en WildFly. En este trabajo no he sido capaz de hacerlo funcionar por lo que esta configuración la he realizado manualmente como se ve en el siguiente apartado.
- httpserver.hostname: aquí definimos el nombre interno de la máquina. Estos nombres son importantes de cara a la emisión de certificados SSL.
- httpserver.dn: aquí definimos el DN del certificado SSL que utilizará el interface WEB de Administración.
- httpserver.external.fqdn: aquí definiremos el nombre del host externo por el que llegaremos a nuestra interface de administración WEB.

4.2.1.1. Personalización keystore.jks

En la instalación realizada y con las versiones descritas todo ha funcionado correctamente en general, con una excepción y es la creación de la keystore.jks para WildFly por parte de EJBCA, la cual nunca conseguí que se generara con la clave definida como parte de web.properties. Por este motivo siguiendo el procedimiento de regeneración de del certificado SSL hemos procedido a su regeneración manual, pero en este caso eligiendo yo y asegurándome de la contraseña correcta. Para ello se sigue el manual descrito en la siguiente página: https://download.primekey.se/docs/EJBCA-Enterprise/latest/SSL_Certificate_Expiration.html (las contraseñas están ocultas por seguridad)

```
bash-4.2$ bin/ejbca.sh ra setendentitystatus tomcat 10
New status for end entity tomcat is 10
bash-4.2$ bin/ejbca.sh ra setclearpwd tomcat XXXXXX
Setting clear text password for user tomcat
bash-4.2$ bin/ejbca.sh batch tomcat
Generating keys in directory /opt/ejbca_ce_7_4_3_2/p12.
Loading configuration from defaults.
Generating RSA keys of size 2048 for tomcat.
Created Keystore for 'tomcat'.
New user generated successfully - tomcat.
bash-4.2$ cd p12/
bash-4.2$ keytool -list -keystore tomcat.jks -storepass
HpSjKcCUGWpPX6A9
Keystore type: jks
Keystore provider: SUN

Your keystore contains 2 entries

cacert, Dec 27, 2021, trustedCertEntry,

Certificate fingerprint (SHA-256):
48:B2:D5:01:1C:6C:3F:E3:EE:A5:F0:9F:4D:DB:E6:70:C0:04:9B:C7:6C:D4:34
:D6:39:27:B5:1A:57:29:0E:A8

localhost, Dec 27, 2021, PrivateKeyEntry,

Certificate fingerprint (SHA-256):
87:01:F7:CA:E7:5D:CA:69:9B:3E:E6:AD:DC:78:48:5B:BE:C0:34:78:95:A5:40
:E4:F2:66:E2:16:0D:49:96:36

Warning:

The JKS keystore uses a proprietary format. It is recommended to
migrate to PKCS12 which is an industry standard format using
"keytool -importkeystore -srckeystore tomcat.jks -destkeystore
tomcat.jks -deststoretype pkcs12".
```

4.2.1.2. Generación del certificado de Super Admin

El ultimo paso que necesitaremos para que nuestra CA sea usable, es generar el certificado mTLS para que el usuario Super Admin se pueda autenticar con la plataforma. Para ello seguimos el mismo mecanismo que hemos descrito en el punto anterior, pero cambiando el usuario tomcat por Super Admin:

```
bash-4.2$ bin/ejbca.sh ra setendentitystatus superadmin 10
New status for end entity superadmin is 10
bash-4.2$ bin/ejbca.sh ra setclearpwd superadmin a123456.
Setting clear text password for user superadmin
bash-4.2$ bin/ejbca.sh batch
Use 'batch --help' for additional options.
Generating keys in directory /opt/ejbca_ce_7_4_3_2/p12.
Generating for end entities with status NEW.
Batch generating 1 users.
Loading configuration from defaults.
Generating RSA keys of size 2048 for superadmin.
Created Keystore for 'superadmin'.
New user generated successfully - superadmin.
1 new users generated successfully - :superadmin.
Generating for end entities with status FAILED.
Batch generating 0 users.
```

Con esto tendremos generado un fichero .p12 en el directorio p12/ de EJBCA que podremos importar en cualquier navegador y hará las veces de mecanismo de autenticación de cliente frente a la plataforma.

A partir de este momento ya tenemos EJBCA configurado correctamente y con una CA de gestión lista para desplegarse, simplemente deberemos seguir los tres pasos necesarios para la construcción, e instalación posterior que consiste en la generación de las keystores y su copia a los directorios protegidos de WildFly.

4.3. ACME

El tercer elemento de la solución es la automatización de la emisión de los certificados a las aplicaciones. El software de EJBCA como ya hemos descrito realizará las tareas de gestión de la PKI, pero el objetivo que buscamos es disponer de la capacidad que nuestras aplicaciones puedan solicitar (bajo las medidas de seguridad que sean necesarias) los certificados necesarios, por ejemplo, para negociaciones mTLS o certificados SSL, para que las aplicaciones puedan utilizarlas como mecanismos de comunicación segura, así como mecanismos de autenticación.

Este elemento será igualmente configurado en la VPC que hemos denominado VPC de trabajo de manera que no será accesible directamente por parte de las cargas de trabajo para su administración, y que expondrá sus servicios vía el servicio ya descrito como

AWS PrivateLink de manera que este mecanismo será el que permitirá el control de acceso a la herramienta ACME directamente en la VPC que se encuentra desplegado.

EJBCA y ACME disponen de un mecanismo de integración nativo que permitirá la gestión y emisión de certificados para las aplicaciones vía una interface de API REST.

Este mecanismo ACME, que actuará como la interface entre la PKI y las aplicaciones que de manera automática tengan que obtener o renovar certificados quedaría dentro del diagrama de arquitectura de la siguiente manera:

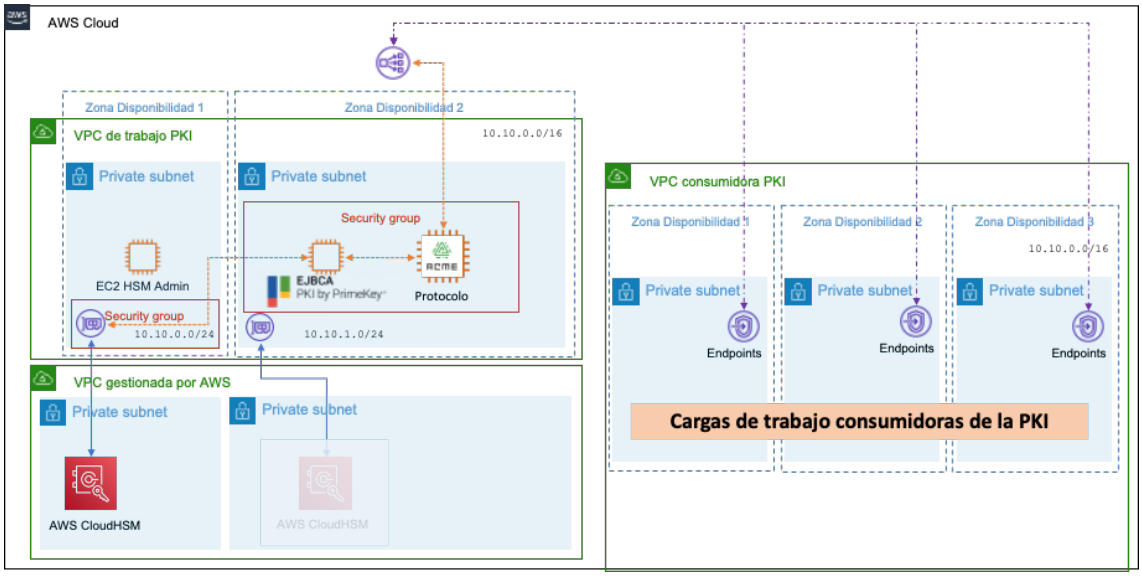


Ilustración 11 - Arquitectura completa AWS de la solución

4.4. CloudHSM

El siguiente paso es el despliegue de nuestra plataforma de CloudHSM que será el dispositivo Hardware que utilizaremos en la nube para almacenar de manera segura nuestras claves privadas. Para ello utilizaremos de la VPC de la PKI sus subredes privadas definidas en nuestra arquitectura.

El primer paso que necesitamos realizar para ello es la creación de un clúster de CloudHSM. Este paso permite la asociación de nuestros futuros dispositivos físicos en una región de AWS, y permite a AWS generar las claves de administración del clúster, que además serán firmadas por nosotros para garantizar la exclusividad criptográfica en el acceso a los servicios del HSM. El dispositivo HSM se asociará a una VPC y determinaremos desde que subredes se podrá acceder a ellos. El clúster como tal, estará conformado por un número determinado (28 como máximo) de dispositivos HSM. En el modelo que se construye en la presente prueba será 1 dispositivo.

El siguiente paso será el despliegue de nuestro primer HSM físico dentro del clúster que hemos creado, así como el proceso de inicialización. Todos estos pasos los realizaremos desde la consola de AWS ya que a día de hoy aun no está soportado el uso de modelos de scripting como CloudFormation para automatizar su despliegue.

El siguiente paso es la inicialización del clúster, el cual en primer lugar nos solicitará el despliegue de nuestro primer dispositivo físico, y para ello deberemos elegir una zona de disponibilidad.

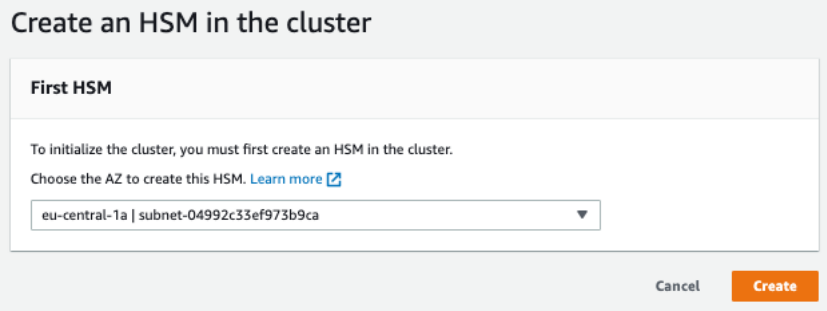


Ilustración 12 - Creación CloudHSM

Una vez desplegado el nuevo hardware, el sistema nos dará el fichero CSR del clúster para que lo firmemos con nuestras propias claves y así garantizar la confidencialidad el sistema. También nos ofrece la posibilidad de descargar el certificado del HSM generado con ese CSR, así como otros pertenecientes a la cadena hasta el del fabricante del Hardware:

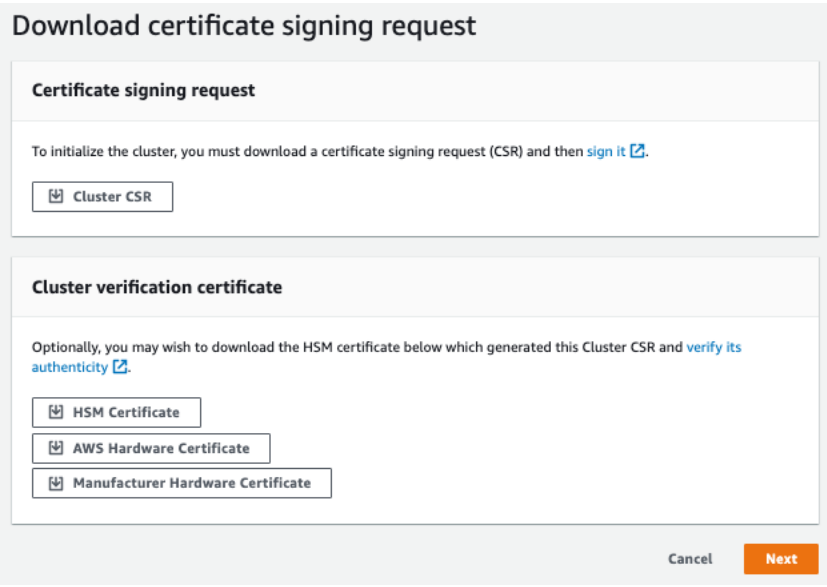


Ilustración 13 - Descarga CSR de CloudHSM

Para ello vamos a generar los certificados y firmar el CSR que nos ha facilitado AWS.

- Generamos la clave privada que vamos a utilizar:

```
[juangp@147dda8b6758 CloudHSM % openssl genrsa -aes256 -out customerCA.key 2048
Generating RSA private key, 2048 bit long modulus
.....................................................+++
........+++
e is 65537 (0x10001)
[Enter pass phrase for customerCA.key:
[Verifying - Enter pass phrase for customerCA.key:
```

Ilustración 14 - Generación clave privada

- Creamos un certificado que vamos a firmar con la clave privada recién generada. Este certificado será con el que nos conectemos a nuestro clúster de dispositivos HSM en el futuro, y que tendremos que guardar para configurarlo en cada máquina que utilicemos el cliente.

```
[juangp@147dda8b6758 CloudHSM % openssl req -new -x509 -days 3652 -key customerCA.key -out customerCA.crt
[Enter pass phrase for customerCA.key:
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) []:ES
State or Province Name (full name) []:Madrid
Locality Name (eg, city) []:Madrid
Organization Name (eg, company) []:juanen
Organizational Unit Name (eg, section) []:UOC
Common Name (eg, fully qualified host name) []:cloudhsm.uoc.aws.juanen.com
Email Address []:juanen@uoc.edu
```

Ilustración 15 - Generación certificado para inicializar CloudHSM

- Ahora vamos a firmar con la misma clave, el CSR de nuestro clúster:

```
juangp@147dda8b6758 CloudHSM % openssl x509 -req -days 3652 -in cluster-g2cszapzdgk_ClusterCsr.csr -CA customerCA.crt
 -CAkey customerCA.key -CAcreateserial -out cluster-g2cszapzdgk_CustomerHsmCertificate.crt
Signature ok
subject=/C=US/ST=CA/O=Cavium/OU=N3FIPS/L=SanJose/CN=HSM:BCAF69B155BAAAF5FCBE71AFA1F788:PARTN:6, for FIPS mode
Getting CA Private Key
Enter pass phrase for customerCA.key:
```

Ilustración 16 - Firma del CSR de CloudHSM

- Para finalizar subimos los dos certificados, el CSR que nos entregó firmado, y el certificado de cliente que hemos creado

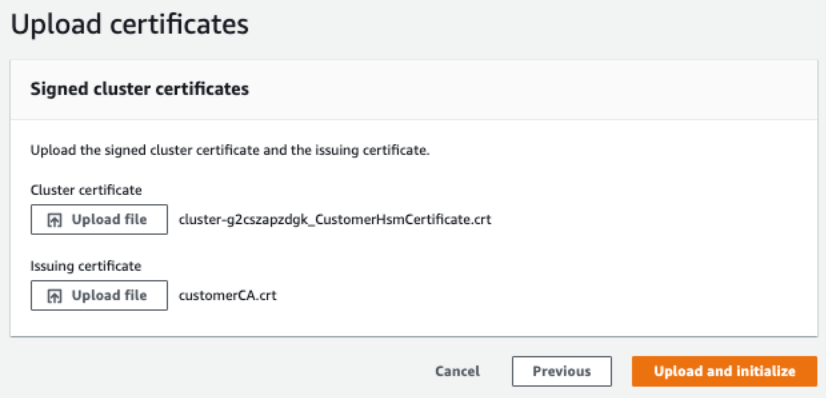


Ilustración 17 - Instalación CSR firmado en CloudHSM

- Una vez termine el proceso de inicialización podremos acceder a CloudHSM con las herramientas de administración para proceder a crear los usuarios que EJBCA utilizará para su utilización.
- Procedemos a descargar el cliente en nuestra máquina de administración (bastión) y configurarlo. El paso más importante es copiar al directorio del cliente /opt/cloudhsm/etc el certificado autofirmado que generamos unos pasos atrás.

- Por último, configuramos el cliente dándole la dirección IP que tiene nuestro CloudHSM y que podremos ver en la consola de AWS:

```
[[root@ip-10-10-10-198 ~]# mv customerCA.crt /opt/cloudhsm/etc/
[[root@ip-10-10-10-198 ~]# sudo /opt/cloudhsm/bin/configure -a 10.10.0.172
Updating server config in /opt/cloudhsm/etc/cloudhsm_client.cfg
Updating server config in /opt/cloudhsm/etc/cloudhsm_mgmt_util.cfg
```

Ilustración 18 - Configuración cliente CloudHSM

- Ahora que tenemos el clúster listo, solo nos queda activarlo eliminando la configuración por defecto para el usuario PRECO.
- No sin antes añadir a la instancia de administración (Bastion) así como a la instancia EJBCA su pertenencia al mismo grupo de seguridad del clúster HSM para permitir su acceso a consumir los servicios.

```
[root@ip-10-10-10-198 ~]# /opt/cloudhsm/bin/cloudhsm_mgmt_util /opt/cloudhsm/etc/cloudhsm_mgmt_util.cfg
Ignoring E2E enable flag in the configuration file

Connecting to the server(s), it may take time
depending on the server(s) load, please wait...

Connecting to server '10.10.0.172': hostname '10.10.0.172', port 2225...
Connected to server '10.10.0.172': hostname '10.10.0.172', port 2225.
E2E enabled on server 0(10.10.0.172)
aws-cloudhsm>listUsers
Users on server 0(10.10.0.172):
Number of users found:2

    User Id            User Type       User Name                      MofnPubKey    LoginFailureCnt        2FA
         1             PRECO           admin                               NO                  0         NO
         2             AU              app_user                            NO                  0         NO


aws-cloudhsm>loginHSM PRECO admin password
loginHSM success on server 0(10.10.0.172)
aws-cloudhsm>changePswd PRECO admin a123456.
*************************CAUTION**********************************
This is a CRITICAL operation, should be done on all nodes in the
cluster. AWS does NOT synchronize these changes automatically with the
nodes on which this operation is not executed or failed, please
ensure this operation is executed on all nodes in the cluster.
******************************************************************

Do you want to continue(y/n)?y
Changing password for admin(PRECO) on 1 nodes
changePswd success on server 0(10.10.0.172)
aws-cloudhsm>listUsers
Users on server 0(10.10.0.172):
Number of users found:2

    User Id            User Type       User Name                      MofnPubKey    LoginFailureCnt        2FA
         1             CO              admin                               NO                  0         NO
         2             AU              app_user                            NO                  0         NO


aws-cloudhsm>
```

Ilustración 19 - Inicialización de usuarios CloudHSM

- En la imagen vemos el proceso y como una vez activado, después de cambiar la contraseña al usuario admin el tipo de usuario PRECO pasa a ser CO (Crypto Officer).

- A continuación, nos volvemos a logar con el usuario admin, pero en esta ocasión crearemos un usuario para la plataforma EJBCA:

```
aws-cloudhsm>login CO admin a123456.
loginHSM success on server 0(10.10.0.172)
aws-cloudhsm>createUser CU ejbca a123456.
**************************CAUTION**********************************
This is a CRITICAL operation, should be done on all nodes in the
cluster. AWS does NOT synchronize these changes automatically with the
nodes on which this operation is not executed or failed, please
ensure this operation is executed on all nodes in the cluster.
*******************************************************************

Do you want to continue(y/n)?y
Creating User ejbca(CU) on 1 nodes
createUser success on server 0(10.10.0.172)
aws-cloudhsm>listUsers
Users on server 0(10.10.0.172):
Number of users found:3

    User Id            User Type       User Name                      MofnPubKey    LoginFailureCnt         2FA
         1             CO              admin                               NO              0              NO
         2             AU              app_user                            NO              0              NO
         3             CU              ejbca                               NO              0              NO
```

Ilustración 20 - Creación usuarios EJBCA en CloudHSM

- Una vez generado el usuario ejbca es hora de probar la integración con EJBCA y CloudHSM creando tres usuarios de prueba.

```
-bash-4.2$ ./p11ng-cli.sh generatekeypair --lib-file
/opt/cloudhsm/lib/libcloudhsm_pkcs11.so --slot-ref SLOT_LABEL --slot
cavium --alias testKey0001 --key-spec RSA2048 --key-usage
SIGN_ENCRYPT
Enter slot login password:

Generated key pair with alias testKey0001
-bash-4.2$ /opt/ejbca/dist/p11ng-cli/p11ng-cli.sh generatekeypair --
lib-file      /opt/cloudhsm/lib/libcloudhsm_pkcs11.so      --slot-ref
SLOT_LABEL --slot cavium --alias signKey0001 --key-spec RSA2048 --
key-usage SIGN_ENCRYPT
Enter slot login password:

Generated key pair with alias signKey0001
-bash-4.2$ /opt/ejbca/dist/p11ng-cli/p11ng-cli.sh generatekeypair --
lib-file      /opt/cloudhsm/lib/libcloudhsm_pkcs11.so      --slot-ref
SLOT_LABEL --slot cavium --alias defaultKey0001 --key-spec RSA2048 -
-key-usage SIGN_ENCRYPT
Enter slot login password:

Generated key pair with alias defaultKey0001
```

- Comprobamos en el HSM que se han creado las claves:

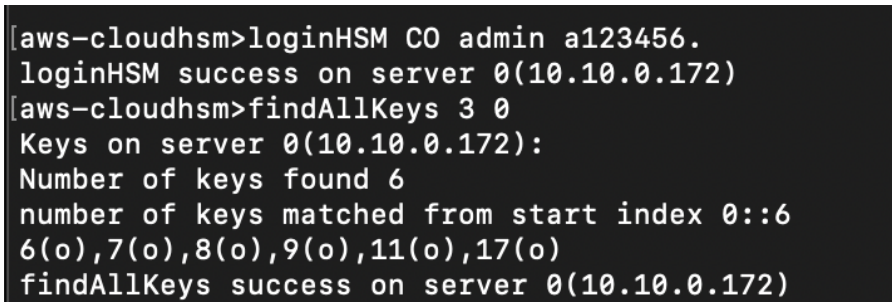


Ilustración 21 - Validación de claves en CloudHSM

Con este último paso, hemos comprobado como se inicializa el clúster de AWS CloudHSM y como en este proceso las claves que debemos usar son generadas por nosotros (para este caso por sencillez hemos utilizado openssl, pero podríamos usar cualquier otro mecanismo más seguro para garantizar la confidencialidad e las claves en el proceso de firma). Adicionalmente, hemos probado la integración de la solución de PKI EJBCA y vemos como se generan claves desde su CLI, y como estas cuando listamos las claves del usuario 3 (usuario ejbca en el AWS CloudHSM) aparecen todas las que tenemos generadas.

4.5. Consumo de los servicios de PKI

Como se puede ver en los diagramas de arquitectura de la solución la que denominamos "VPC consumidora PKI" será donde nuestros clientes de la PKI desplegarán sus cargas de trabajo. En el trabajo que se está realizando el presente documento desplegaremos una arquitectura de microservicios sencilla para demostrar la funcionalidad buscada en los objetivos de este documento.

Esta arquitectura será una página web construida en un micro servicio, la cual será expuesta a internet mediante el protocolo HTTPS y su correspondiente balanceador.

Uno de los casos de uso que contemplamos, es la posibilidad de realizar la encriptación del tráfico en transito entre los diferentes microservicios desplegados, utilizando TLS y apoyado en la configuración de un Service Mesh que definirá esas políticas.

En este caso desplegaremos certificados de servidor, de manera que un micro servicio pueda comunicarse de manera segura con otro que realiza la función de servidor con este mecanismo de TLS usando certificados confiables que han sido emitidos para ese propósito por la PKI.

Con objeto de evitar el uso de piezas muy especificas de AWS y hacer el presente trabajo lo más extensivo posible para cualquier nube, e incluso sistemas on-premises utilizaremos el servicio de Kubernetes gestionado de AWS, el cual es 100% compatible con la versión downstream de la distribución opensource de Kubernetes. Esto nos permitirá utilizar una segunda pieza que será la responsable de orquestar las políticas que los microservicios requerirán para poder hablar entre si.

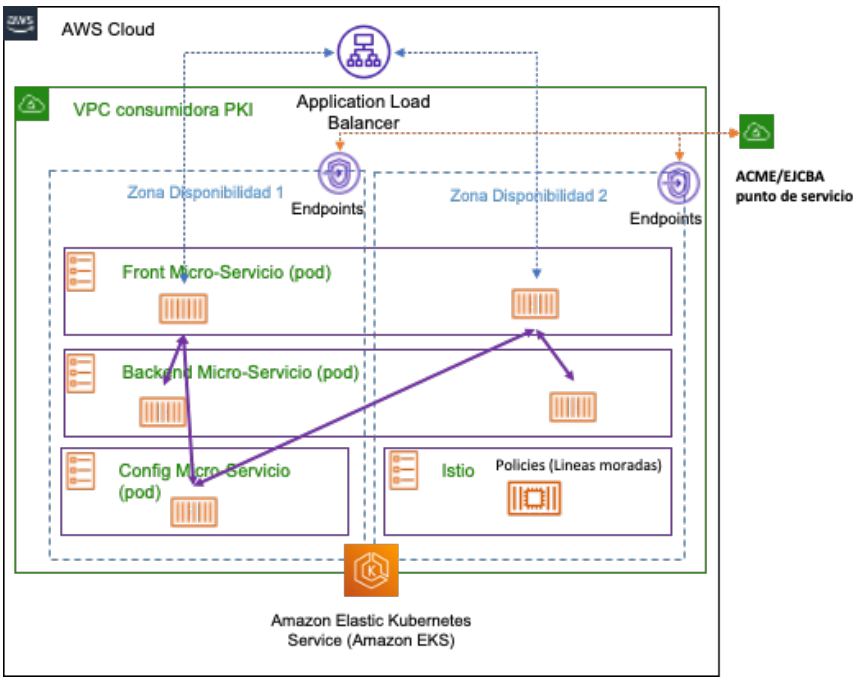


Ilustración 22 - Arquitectura de microservicios integrada con ACME

Como se ve en la imagen, desplegaremos los dos micro servicios sencillos que construyen la web, y un tercer micro servicio que será el responsable de entregar las configuraciones a estos primeros. Para poder hablar los microservicios entre ellos se deberán autenticar utilizando mTLS (Front con Back y ambos con el micro servicio de Config), y esto solo podrán hacerlo si desde ACME han conseguido que se les entregue el certificado TLS de manera automática.

La gestión de estas políticas se hará utilizando AppMesh que es una herramienta de Service Mesh adoptada en el mercado de los contenedores en general y en el mundo Kubernetes para AWS en particular y que nos ayudará a publicar esas políticas que fuercen que los micro servicios solo puedan hablar si se autentican con ese mecanismo TLS.

4.6. Despliegue de la aplicación consumidora

Ya está instalada la CA basada en la solución EJBCA, así como configurado el sistema de almacenamiento seguro de certificados privados basado en CloudHSM así como configurado el servicio ACME como parte de EJBCA. Estos servicios son consumibles vía un balanceador que se expone externamente a la VPC de la PKI. Al igual que si fuéramos una aplicación empresarial tendremos nuestro propio espacio en forma de VPC y tendremos desplegado en él una aplicación de ejemplo llamada Yelb (https://github.com/mreferre/yelb) que será la que utilizaremos como base para demostrar el objetivo del presente trabajo.
Para ello vamos a seguir las instrucciones que la propia aplicación describe de manera que desplegaremos las capacidades de infraestructura que necesitamos y posteriormente desplegaremos la propia aplicación. La arquitectura de la aplicación es la siguiente:

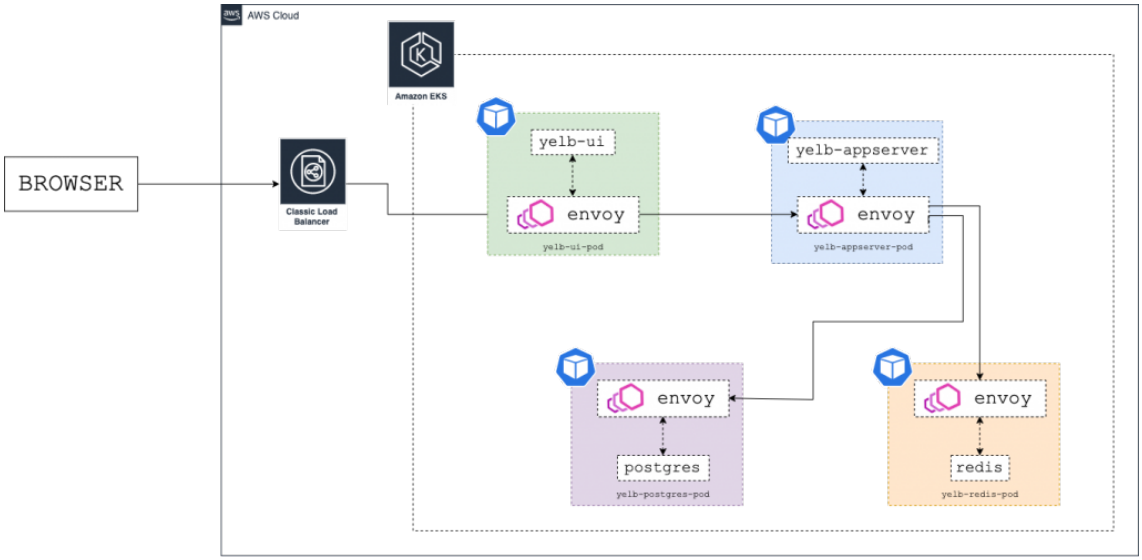


Ilustración 23 - Arquitectura aplicación de demo

(fuente https://aws.amazon.com/blogs/containers/getting-started-with-app-mesh-and-eks/)

Dentro de nuestro trabajo queremos proteger esta aplicación con una PKI interna, que haga que la conversación entre los diferentes microservicios que se ven en la arquitectura ocurra encriptada usando TLS. Para ello vamos a necesitar que EJBCA (nuestra plataforma de PKI) sea capaz de entregar estos certificados TLS automáticamente a los diferentes micro servicios.
Como se puede ver esta es una plataforma basada en Kubernetes y para poder gestionar las políticas que fuercen a los microservicios a hablar de manera encriptada usando TLS vamos a utilizar una de las capacidades que ofrece Kubernetes denominada Service Mesh. Existen diferentes soluciones de Service Mesh, como puede ser Isitio que es una de las más adoptadas. Por sencillez en este trabajo vamos a optar por utilizar AppMesh que es el servicio nativo de AWS pero el procedimiento y modo de despliegue sería muy similar. AWS AppMesh nos va a permitir crear esas políticas que obligan que el tráfico entre microservicios sea TLS, y para ello lo que hará es desplegar lo que se conoce como un side-car. Esto no es más que un mecanismo que se pone delante del microservicio o contenedor para forzar que el tráfico pase por él y ahí es donde se forzarán las diferentes políticas.

Primero añadiremos el repositorio de paquetes que necesitamos para instalar AppMesh usando el gestor de paquetes helm, para a continuación crear la partición (namespace en

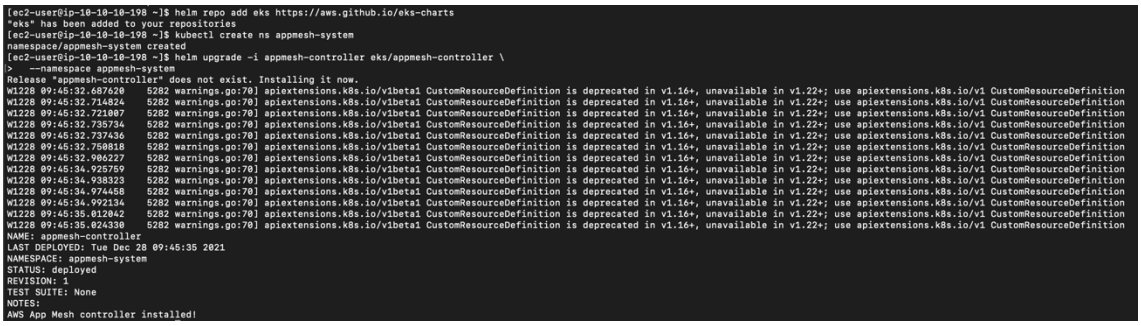


Ilustración 24 - Instalación del Service Mesh

Kubernetes) para instalar de manera independiente AppMesh de la aplicación de producción que queremos proteger.

Tras esto, simplemente nos quedará mover nuestra aplicación Yelb al namespace nuevo, y permitir que AppMesh pueda inyectar esas políticas a través de los side-car.

Para que AppMesh pueda gestionar y controlar los servicios que tenemos en Kubernetes, tenemos que crear lo que se llama un servicio virtual. Este servicio virtual no será otra cosa que un Pod con dos contenedores, el contenedor original con el contenedor que va a gestionar las políticas. Este contenedor no es otra cosa que un proxy llamado Envoy. Como los POD ya están desplegados, una vez que definamos los nuevos servicios virtuales mataremos todos los POD existentes para que se redesplieguen bajo el gobierno de AppMesh.

En la imagen podemos ver como en cada POD hay dos contenedores corriendo ahora.

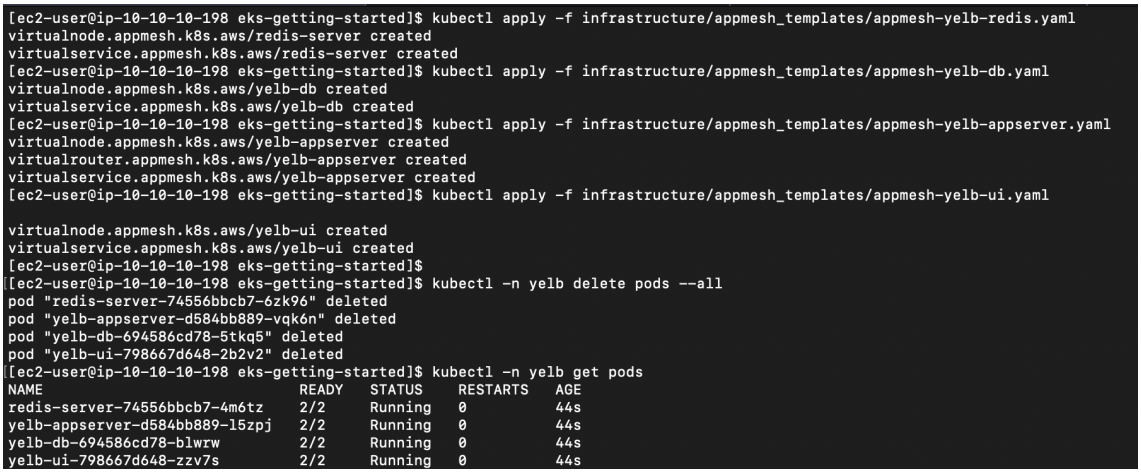


Ilustración 25 - Aplicación de demo con Service Mesh

4.6.1. Instalación y configuración del gestor de certificados

Ahora que nuestra aplicación está funcionando es el momento de configurar la solución de gestión de certificados. Cert-Manager es una solución nativa de Kubernetes que nos permite esta gestión de certificados (emisión) y que es compatible con diferentes mecanismos, desde la auto-emisión de certificados dándole una clave privada y pública

que hayamos generado con por ejemplo OpenSSL, o integrándolo con protocolos como ACME con PKIs existentes ya sea reconocidas, o privadas como es nuestro caso con EJBCA.

Por lo tanto, lo primero que haremos es crear un namespace para el servicio y lo instalaremos. Una vez creado, podremos comprobar que está funcionando:

```
[ec2-user@ip-10-10-10-198 eks-getting-started]$ kubectl -n cert-manager get pods
NAME                                       READY   STATUS              RESTARTS   AGE
cert-manager-cainjector-fc6c787db-6ltk2    0/1     ContainerCreating   0          6s
cert-manager-d994d94d7-vv2r5               0/1     ContainerCreating   0          6s
cert-manager-webhook-5c5ddb797c-7pqht      0/1     ContainerCreating   0          6s
```

Ilustración 26 - Instalación de cert-manager.io

En este momento debemos tener disponible el endpoint que expone los servicios de EJBCA como PKI en la VPC de nuestra aplicación. Con esta información configuraremos nuestro cert-manager para que solicite vía ACME los certificados a EJBCA.

```yaml
apiVersion: v1
kind: Service
metadata:
  name: pki-service
  namespace: cert-manager
spec:
  ports:
  - port: 443
    targetPort: 8442

---
apiVersion: v1
kind: Endpoints
metadata:
  name: pki-service
  namespace: cert-manager
subsets:
- addresses:
  - ip: 10.192.20.9
  ports:
  - port: 8442
```

Ilustración 27 - Configuracion de cert-manager con ACME

Una vez desplegado, solo nos queda descargar el certificado de la CA configurada en la PKI para hacer que cert-manager pueda confiar en él:

```
apiVersion: v1
kind: Secret
metadata:
  namespace: cert-manager
  name: secret-acme-tls-ca
type: Opaque
stringData:
  AcmeTlsEndPointIssuingCa.crt: |+
    -----BEGIN CERTIFICATE-----
    MIIE3TCCA0WgAwIBAgIUCr4fLGbFLWy5XNivvcbOCYOE7HowDQYJKoZIhvcNAQEL
    BQAwdjEVMBMGA1UEAwwMTWFuYWdlbWVudENBMT0wOwYDVQQLDDRlYzItMTgtMTg0
    LTExLTI0NC5ldS1jZW50cmFsLTEuY29tcHV0ZS5hbWF6b25hd3MuY29tMR4wHAYD
    VQQKDBVhbWktMDAzMDA3NmIwNDVhMTQ4ZGYwHhcNMjExMjI3MTkxMjQxWhcNNDYx
    MjI4MTkxMjQwWjB2MRUwEwYDVQQDDAxNYW5hZ2VtZW50Q0ExPTA7BgNVBAsMNGVj
    Mi0xOC0xODQtMTEtMjQ0LmV1LWNlbnRyYWwtMS5jb21wdXRlLmFtYXpvbmF3cy5j
    b20xHjAcBgNVBAoMFWFtaS0wMDMwMDc2YjA0NWExNDhkZjCCAaIwDQYJKoZIhvcN
    AQEBBQADggGPADCCAYoCggGBAJgyWUKX4JogFWL8nu/fIQSg55rNrgBstDkvO8VV
    KCU5rwGsRff68G2diZjt4E7IpTyDmwFXjP08CwaKP0RJJeGEwyc6CYmzAhZgRnWM
    gc+3nCaPTUANg6Sq+yFMh99x/gnry7qKs4p3Ub8jcIF0VaB4J6bAfgbiJhnZFBv9
    +mUQx9CzibcIqyWMkloV5GEn5lw3/0i1+vXBfxXoJqu+iBTYtsPFBT/43RrXgJ6F
    S1nzSBsoMc81RjcRoEfQ/oCzIrn78g4Svbds7PLTu65VEG0rTN1hWIpsEQhuIR3Y
    r+c/rdDLmsmcmm9cmEM+Gr/bAXT6b90tZfx1WDct5OSMjqNgWwHbU9FlRSN+c8ha
    ECmF/c5MEVko3uvwzAXVZeRa/0MXbX7/2w84e9g58u0bNS/hq1hVJw+Y5VwBAhbd
    kaBTVukYYThReSFDM1n5fBbWsCb9DRRZc2IpF8uFIeTUppW7lrA5E/7bMapVmjHB
    koboC9j2pUST5NbY4dsT0vayTQIDAQABo2MwYTAPBgNVHRMBAf8EBTADAQH/MB8G
    A1UdIwQYMBaAFIwWj2RfkCGMMlxIBzxvNSTOHyiQMB0GA1UdDgQWBBSMFo9kX5Ah
    jDJcSAc8bzUkzh8okDAOBgNVHQ8BAf8EBAMCAYYwDQYJKoZIhvcNAQELBQADggGB
    AFFdxxmLE3ZCDDnYb7LHd6RjcBQoumI4lfLvktgR2z1ASCHmpWzK3TDkDE10fs4O
    OmnzcTS0xUe8ouWfIpBXDWCJAfthBpDNeScDgxIx0wn9K1MIj5q2R40YUURe5sOV
    z6We6Yyvl8N39TJ4LIYK34pcMdKRi6RH4xKW/SAqZufsIkK7KIE77Xu4zmjv9JmX
    AteYqPX71ofjzHfMEbqIdllXQHEj6Abl4oEaErjD1PMSYWQ4WMvfHKZAwxyWPIqM
    L27PaHOCuQ4bzzxkco61Eo8ZqTb72gPgyhZmS4TDx1XiIWqeKsSiU1EuU6XK6yXK
    vqxzFBrZyW4ZTrRa1h+DpfpuA2NrZI0jBm2Owmk6pyxFY4mlG2WKwoJQXax140Tg
    kDEAy5HnjNWCjtN+Jc7XXe2svdSN6BVJrPWsE3KNyS3t0jjMITxLZfcYEwg8Vefz
    Ast0eK5Qr7s14KCI/pprY7xRAWfz8Wm30RnB7FNNSjXoAPmNHQ5mtYrWKTcUBeAH
    mg==
    -----END CERTIFICATE-----
```

Ilustración 28 - Instalación del CRT de EJBCA

Con ello estamos listos para realizar la conexión entre cert-manager y el servicio ACME de EJBCA. Lanzamos el servicio llamado CertificateIssuer que en primer lugar hará el registro con ACME:

```
[[ec2-user@ip-10-10-10-198 eks-getting-started]$ kubectl describe clusterissuer clusterissuer-ejbca-acme
Name:         clusterissuer-ejbca-acme
Namespace:
Labels:       <none>
Annotations:  <none>
API Version:  cert-manager.io/v1
Kind:         ClusterIssuer
Metadata:
  Creation Timestamp:  2021-12-28T14:52:45Z
  Generation:          1
  Managed Fields:
    API Version:  cert-manager.io/v1
    Fields Type:  FieldsV1
    fieldsV1:
      f:status:
        .:
        f:acme:
          .:
          f:lastRegisteredEmail:
          f:uri:
        f:conditions:
    Manager:      controller
    Operation:    Update
    Time:         2021-12-28T14:52:45Z
    API Version:  cert-manager.io/v1
    Fields Type:  FieldsV1
    fieldsV1:
      f:spec:
        .:
        f:acme:
          .:
          f:email:
          f:privateKeySecretRef:
            .:
            f:name:
          f:server:
          f:solvers:
    Manager:         kubectl-create
    Operation:       Update
    Time:            2021-12-28T14:52:45Z
  Resource Version:  73714
  UID:               09242aba-550c-4979-97a5-143d6bf1a692
Spec:
  Acme:
    Email:            juanen@uoc.edu
    Preferred Chain:
    Private Key Secret Ref:
      Name:  issuer-ejbca-acme-secret
    Server:  https://ec2-18-184-11-244.eu-central-1.compute.amazonaws.com/ejbca/acme/directory
    Solvers:
      http01:
        Ingress:
          Class:  nginx
      Selector:
Status:
  Acme:
    Last Registered Email:  juanen@uoc.edu
    Uri:                    https://ec2-18-184-11-244.eu-central-1.compute.amazonaws.com/ejbca/acme/acct/MSn1GbN4gK4ay5jyY_qc7w
  Conditions:
    Last Transition Time:  2021-12-28T14:52:45Z
    Message:               The ACME account was registered with the ACME server
    Observed Generation:   1
    Reason:                ACMEAccountRegistered
    Status:                True
    Type:                  Ready
Events:                    <none>
```

Ilustración 29 - Integración cert-manager y ACME finalizada

Como se puede observar en la imagen superior el sistema está registrado con el servidor ACME. Tenemos otro mecanismo denominado Issuer dentro de cert-manager, pero ClusterIssuer es más cómodo ya que funciona a nivel de clúster, y desde cualquier namespace podremos solicitar certificados. Es cierto que puede complicar un poco los mecanismos de validación de cert-manager para la emisión de certificados, pero una vez configurado ofrece muchas ventajas. (Cryptosense, 2021)

A partir de este momento deberíamos ser capaces de solicitar certificados al servidor de la siguiente manera:

```yaml
apiVersion: cert-manager.io/v1
kind: Certificate
metadata:
  name: cert-testsystem-example-org
  namespace: default
spec:
  secretName: cert-secret-testsystem-example-org
  issuerRef:
    name: clusterissuer-ejbca-acme
    kind: ClusterIssuer
  commonName: testsystem.example.org
  dnsNames:
  - testsystem.example.org
```

Ilustración 30 - Creación vía ACME de un certificado

Si todo ha ido correcto veremos nuestro certificado emitido:

```
Events:
  Type    Reason     Age   From          Message
  ----    ------     ----  ----          -------
  Normal  Issuing    19s   cert-manager  Issuing certificate as Secret does not exist
  Normal  Generated  19s   cert-manager  Stored new private key in temporary Secret resource "cert-testsystem-example-org-jf6rb"
  Normal  Requested  19s   cert-manager  Created new CertificateRequest resource "cert-testsystem-example-org-xwnfk"
```

Ilustración 31 - Emisión del certificado vía ACME

Sin embargo, cuando comprobamos el estado de emisión de los certificados tenemos que ver en detalle como funciona la integración del protocolo ACME de cert-manager para la obtención de los certificados firmados. ACME utiliza un mecanismo que "orders" o peticiones para la solicitud de los certificados, que podemos ver en la siguiente imagen (solo se muestra la información relevante):

Comando: `kubectl describe certificaterequests cert-testsystem-org`

```
Status:
  Conditions:
    Last Transition Time:  2021-12-28T21:54:25Z
    Message:               Certificate request has been approved by cert-manager.io
    Reason:                cert-manager.io
    Status:                True
    Type:                  Approved
    Last Transition Time:  2021-12-28T21:54:25Z
    Message:               Waiting on certificate issuance from order default/cert-testsystem-example-org-pp5cw-3107055218: "pending"
    Reason:                Pending
    Status:                False
    Type:                  Ready
Events:
  Type    Reason           Age   From          Message
  ----    ------           ----  ----          -------
  Normal  cert-manager.io  12s   cert-manager  Certificate request has been approved by cert-manager.io
  Normal  OrderCreated     12s   cert-manager  Created Order resource default/cert-testsystem-example-org-pp5cw-3107055218
```

Ilustración 32 - Datos de la orden de cert-manager.io

La orden hace el seguimiento de como se lanza la petición de un certificado. El certificado se compone de un CSR que es lo que compone esta orden y que cert-manager denomina CertificateRequest. El punto interesante que encontramos, es que esta orden, dependiendo del número de dominios para los que se solicite va a generar lo que denomina un reto o challenge que debe ser resuelto para demostrar que el dominio está bajo nuestro control.

Este challenge puede ser de dos tipos, bien DNS01 o bien HTTP01. En el primer caso lo que hará cert-manager es crear un registro de tipo TXT con este formato: _acme-challenge.ejemplo.com con un token. De esta manera validará que somos dueños del dominio y procederá a emitir el certificado. En el caso de utilizar el protocolo HTTP01 este lo que hará es crear un contenedor que contiene ese token que espera, y modificar el mecanismo de ingress del clúster para redirigirlo al mismo y validar así el dominio. De esta manera es imprescindible que el dominio que utilicemos sea resoluble dentro de nuestro clúster de Kubernetes o nunca ocurrirá la validación.

Con este último paso tan solo nos restará definir los certificados que queremos que sean emitidos para nuestros servicios. Volviendo al ejemplo que se ha utilizado para demostrar esta integración con la aplicación Yelb, por ejemplo, este sería el que utilizaríamos para uno de estos servicios:

```yaml
apiVersion: cert-manager.io/v1
kind: Certificate
metadata:
  name: yelb-cert-db
  namespace: yelb
spec:
  dnsNames:
    - "yelb-db.yelb.svc.cluster.local"
  secretName: yelb-tls-db
  issuerRef:
    name: clusterissuer-ejbca-acme
    kind: ClusterIssuer
```

Ilustración 33 - Configuración certificados para aplicación de demo

Una vez configurados todos los certificados, si pedimos su status obtendremos algo similar a esto:

```
[ec2-user@ip-10-10-10-198 ~]$ kubectl -n yelb get cert -o wide

NAME              READY   SECRET            ISSUER
STATUS                                                  AGE
yelb-cert-app     True    yelb-tls-app      clusterissuer-ejbca-acme
Certificate is up to date and has not expired   4m23s
yelb-cert-db      True    yelb-tls-db       clusterissuer-ejbca-acme
Certificate is up to date and has not expired   4m23s
yelb-cert-gw      True    yelb-tls-gw       clusterissuer-ejbca-acme
Certificate is up to date and has not expired   4m23s
yelb-cert-redis   True    yelb-tls-redis    clusterissuer-ejbca-acme
Certificate is up to date and has not expired   4m23s
yelb-cert-ui      True    yelb-tls-ui       clusterissuer-ejbca-acme
Certificate is up to date and has not expired   4m23s
```

Comprobando de esta manera que todos los microservicios han obtenido su certificado de manera automática y que ahora podremos configurar dentro de AWS AppMesh como obligatorios para los diferentes microservicios que componen nuestra aplicación.

5. Conclusiones

Los objetivos que planteábamos al inicio de este trabajo se centraban en el análisis de los servicios criptográficos en la nube pública, demostrar la viabilidad de utilizar estándares en la industria, por ejemplo, el protocolo ACME, o analizar la compatibilidad de sistemas tradicionales o de nube pública.

5.1. Servicios de criptografía en la nube pública

La realidad de los servicios de criptografía de nube pública es que es muy variada en función de los proveedores que utilicemos. Existe un patrón común en todos ellos y que trata de facilitar el uso de estos en cualquier circunstancia, integrando estos servicios con sus servicios de computación en la nube de manera nativa reduciendo al máximo cualquier complejidad. Todos los proveedores tienen servicios con un nombre común, Key Management Service, que no es otra cosa que una partición de un módulo de hardware criptográfico (HSM) de alto rendimiento que será capaz de realizar todas las operaciones de encriptación y des-encriptación, así como las necesarias de generación de todo tipo de claves, simétricas o asimétricas y diferentes algoritmos.

Todos los servicios están respaldados por hardware físico y en general con las máximas certificaciones del tipo FIPS 140-2 nivel 2 (o superior). Esto es algo clave para estos proveedores de nube pública ya que buscan ofrecer a sus clientes la máxima seguridad, incluido los que los entornos regulados que puedan requerir.

Desde un punto de operación estos servicios ofrecen todas las operaciones criptográficas esperables, y dependiendo del proveedor gestionan las situaciones de maneras ligeramente diferentes. Por ejemplo, las soluciones de Microsoft utilizan un algoritmo asimétrico para proteger las claves de datos, sin embargo, las soluciones de AWS utilizan algoritmos simétricos para la protección de las claves de datos.

Todos ellos están altamente integrados con la mayoría de los servicios que ofrecen estos proveedores de nube, de manera que almacenar datos encriptados ya sea en una máquina virtual, una base de datos, o un bus de integración es una tarea que en general solo conlleva hacer un click. Es más, en muchísimos casos el uso de estos servicios criptográficos, a pesar de la altísima carga computacional que suele implicar son gratuitos y no tienen coste para el cliente. Esto tiene un contrapunto, y es que es un servicio gestionado, y como tal nuestras capacidades de acceso a claves o de auditoría están limitadas, acciones como las que tenemos en el mundo tradicional en que gestionamos todos los protocolos de los sistemas criptográficos, la operación, el mantenimiento, o los procedimientos, etcétera, aquí tenemos que compartirlos con el proveedor de nube pública. Se utiliza lo que ellos llaman un modelo de responsabilidad compartida de manera que vamos a ceder parte de esa responsabilidad de gestión de nuestros sistemas a un tercero y esto en ocasiones es una situación difícil de integrar en nuestros procesos diarios.

Pero en muchas ocasiones estos servicios no son suficientes, ya que solo ofrecen operaciones generalistas, o nuestros requisitos de cumplimiento exigen disponer por ejemplo de hardware dedicado. Los tres proveedores ofrecen módulos HSM dedicados de diferentes maneras. El proveedor más completo en este sentido es Microsoft ya que ofrece módulos de un tercero que directamente conecta a nuestra nube. En el caso de AWS hace algo similar con módulos hardware de un tercero, pero con un firmware modificado que permite algunas integraciones de este hardware del tercero con servicios

de AWS como es AWS KMS. Por último, la oferta de Google es muy limitada en este aspecto ya que no ofrece acceso directo a este hardware, lo hace a través de su servicio gestionado Google KMS. Igualmente, estos tres hardware, a excepción del de Microsoft son generalistas y tienen carencias en operaciones especializadas de industria, como pueden ser las ceremonias de claves, o métodos por ejemplo en el mundo de medios de pago para traslación de claves o pines que no están disponibles.

En conclusión, la oferta criptográfica es muy interesante en la nube pública, y fomenta la adopción de "encripta todo" fundamentalmente en lo que se refiere a encriptación en reposo. Estos proveedores también complementan la oferta con capacidades de PKI y encriptación en transito mediante certificados SSL, pero lo veremos en el siguiente apartado. Sin embargo, si lo que necesitamos es construir una solución especializada más allá de las operaciones tradicionales de criptografía, tendremos problemas para conseguir estas capacidades por parte de los proveedores de nube. Así mismo, si nuestros procesos son los de una autoridad certificadora también encontraremos algunas limitaciones por ejemplo en las ceremonias de clave debido al modelo de responsabilidad compartida, ya que por ejemplo no tendremos acceso al hardware de ninguna manera, ni si quiera a sus API.

5.2. Viabilidad del uso de estándares en la nube pública

El segundo objetivo que nos marcábamos en el presente trabajo es demostrar que podemos utilizar mecanismos o arquitecturas tradicionales usando servicios criptográficos de nube pública.

La demostración técnica que hemos descrito en el presente ejercicio se ha centrado en la utilización de servicios nativos de nube pública en forma de HSM (AWS CloudHSM) para almacenar las claves privadas, mientras usábamos software bastante estándar en la industria para realizar su integración con el mismos (EJBCA) y lo complementábamos con un protocolo para automatizar la emisión de certificados (ACME). Montar esta arquitectura ha sido bastante complejo en general por el gran número de servicios disponibles, funcionalidades, tipos de certificados, mecanismos de seguridad, integraciones, etc. Y si además le añadimos todas las consideraciones de seguridad que debe tener una PKI lo hacen realmente complejo.

La nube pública tiene un valor clave y que debe estar presente para aprovechar al máximo sus capacidades y es agilidad. Desde luego este trabajo ha demostrado que es muy cuestionable el desplegar una infraestructura de este tipo según el caso de uso. En nuestro caso solo buscábamos desplegar certificados SSL/TLS para nuestros microservicios y preparar esta infraestructura ha sido un trabajo de casi dos semanas, lo que obviamente es altísimo en horas y esfuerzo para algo tan sencillo. Si lo que buscamos es tener una PKI con todos los servicios que podamos necesitar de emisión de certificados de diferentes tipos, entonces la solución no puede ser más satisfactoria. Hemos podido integrar utilizando PKCS#11 la solución de CloudHSM con EJBCA como software de PKI, y somos capaces de gestionar todo lo que necesitamos, desde la autoridad certificadora raíz que creamos en la instalación, así como otros niveles de autoridades intermedias hasta la emisión de certificados finales. Todo ello complementado con servicios como CRL o OCSP que son imprescindibles para cualquier autoridad certificadora.

Hemos encontrado algunas limitaciones que son difíciles de salvar, o al menos hacen que la arquitectura se vuelva compleja. Por ejemplo, una de las propuestas era exponer los

servicios de la PKI internamente a las diferentes cargas de trabajo que puedan necesitar consumirlo de manera interna, aprovechando un mecanismo que AWS denomina VPC PrivateLink. El problema que tendremos que resolver es que necesitaremos construir un mecanismo para tener un Endpoint (fqdn) idéntico en todas las cargas de trabajo que puedan resolver esos VPC PrivateLink de AWS o tendremos problemas con los certificados TLS que EJBCA presenta cuando se le hace cualquier petición. Por supuesto, los Endpoints que utiliza AWS con sus propios nombres de dominio no nos servirían ya que el CN será completamente distinto.

Estas complejidades, así como otras que hemos encontrado en el proceso, dificultan la construcción de una arquitectura segura y nativa siguiendo las buenas prácticas de AWS. Esto no quiere decir que no sea realizable de forma segura, pero nos tendremos que ir a mecanismos de arquitectura más tradicionales y no podremos usar servicios nativos como AWS VPC PrivateLink.

Como comentábamos, el caso de uso será clave en nuestra toma de decisiones, si lo que buscamos no es más que unos microservicios hablen entre si con certificados SSL/TLS que hemos emitido en nuestra autoridad certificadora, desde luego el esfuerzo de desplegar esta infraestructura no lo compensa, teniendo servicios como es el caso de AWS Certificate Manager que es capaz de gestionar el ciclo de vida end-to-end tanto de certificados públicos como nuestra propia PKI interna. Si por el contrario buscamos una infraestructura de PKI más completa y que ofrezca más funcionalidades en la emisión de certificados sin duda es una muy buena aproximación que cubre muchísimos de los casos de uso y añade algunas simplificaciones como puede ser la gestión y operación de dispositivos HSM o la escalabilidad de la plataforma una vez construida.

5.3. Compatibilidad entre sistemas tradicionales y nube pública

Cuando hablamos de nube pública, un tema común entre todos los clientes en general, pero clientes regulados en particular, es no construir arquitecturas que me aten a un proveedor concreto, y que en un futuro potencialmente me permitan moverme a otro proveedor o a mi propio centro de datos.

Si miramos los servicios criptográficos que hemos utilizado aquí, así como los algoritmos usados, podemos concluir que son totalmente compatibles. Concretamente en el caso de AWS CloudHSM es un HSM de industria, que ofrece las librerías y estándares que podemos esperar como PKCS#11 y que nos ofrece mecanismos de transporte de claves en el caso que necesitemos trasladarlas a otra ubicación. No es el caso de los servicios gestionados que describíamos en el anterior punto, por ejemplo, el uso de KMS o Certificate Manager en cualquiera de los proveedores de nube implica que las claves maestras generadas en general no son exportables, por lo que no podremos utilizarlas para llevárnoslas a otro proveedor de nube o a nuestro entro de datos.

Si ponemos nuestros ojos en otras partes de la solución de PKI, como EJBCA o el protocolo ACME, es cierto que la configuración y la base de datos son elementos sencillos de mover, o incluso si hiciéramos copias de las máquinas virtuales. Pero la arquitectura como hemos comentado en el primer apartado de estas conclusiones será realmente compleja por las dependencias de servicios nativos de la nube que es donde encontramos el máximo valor (VPC PrivateLink, mecanismos de seguridad como Security Groups, etc.).

Por supuesto, si hablamos de capacidades criptográficas más especializadas, aquí es donde empezamos a encontrar problemas mayores ya que en general los módulos criptográficos ofrecidos son de tipo generalista y no suelen soportar este tipo de

operaciones especializadas (por ejemplo, como las que encontramos en la industria de medios de pago).

Si vemos la compatibilidad desde un punto de vista de integración y no solamente de "transportabilidad" si encontramos otras consideraciones. Mirando a la arquitectura que se ha desplegado como parte de este trabajo podríamos tener piezas como el software de la PKI en la nube, y los módulos criptográficos on-premise o viceversa sin demasiadas complicaciones técnicas más allá de las comunicaciones.

5.4. Cierre

Se ha mostrado técnicamente como es posible la utilización de servicios nativos de la nube de AWS como son CloudHSM, balanceadores, o VPC Endpoints para exponer los servicios internamente para la construcción y consumo de una PKI. Todos los objetivos se han conseguido salvando los obstáculos que se han ido encontrando en el camino como los descritos en estas conclusiones.

Funcionalmente y técnicamente es viable la construcción de una PKI en la nube pública usando servicios nativos, así como arquitecturas nativas. Aun falta un amplio recorrido de estas herramientas para integrarse aun más con estos servicios nativos que ofrecen los proveedores de nube como es el caso de AWS Certificate Manager con EJBCA el cual podría enriquecer sus capacidades con la emisión de certificados de otros propósitos no soportados actualmente.

La interoperabilidad entre sistemas desplegados en diferentes nubes, así como entre sistemas en nuestros centros de datos está garantizada y solo tenemos que validar el riesgo de la limitación funcional de los productos de nube para ciertos casos de uso.

En definitiva, el uso de servicios de nube pública ha traído algunas ventajas muy interesantes que destaco a continuación:

- Disponer de un hardware HSM como base para nuestras claves privadas, que hemos desplegado en pocos minutos.
- Una vez construida la PKI la capacidad de ofrecer servicios a diferentes cargas de trabajo en AWS usando servicios nativos como VPC PrivateLink
- Aprovecharnos de la capacidad de auto escalado de AWS para escalar nuestra PKI si así fuera necesario (no ha sido parte de este trabajo)

Los siguientes pasos que deberían analizarse es como aprovechar aun más las capacidades disponibles para hacer más sencillo aun que cualquier compañía pueda disponer de una PKI en la nube pública totalmente gestionada e integrada con servicios nativos de nube que aportarán mayor seguridad, escalabilidad y confianza.

6. Bibliografía

Needham, M. (13 de May de 2021). *Worldwide Public Cloud Services Market*. Obtenido de IDC: https://www.idc.com/getdoc.jsp?containerId=prUS47685521

Wikipedia. (13 de Nov de 2021). *Certificate revocation list*. Obtenido de Wikipedia: https://en.wikipedia.org/wiki/Certificate_revocation_list

Amazon Web Services. (11 de October de 2021). *AWS CloudHSM*. Obtenido de AWS CloudHSM: https://aws.amazon.com/es/cloudhsm/

Google Cloud Services. (11 de October de 2021). *Cloud Key Management*. Obtenido de Google Cloud Services: https://cloud.google.com/security-key-management

UK National Cyber Security Centre. (3 de October de 2021). *Design and build a privately hosted Public Key Infrastructure*. Obtenido de National Cyber Security Centre: https://www.ncsc.gov.uk/collection/in-house-public-key-infrastructure/introduction-to-public-key-infrastructure/components-of-a-pki

Amazon Web Services. (16 de Nov de 2021). *AWS CloudHSM Supported Mechanisms*. Obtenido de Amazon Web Services: https://docs.aws.amazon.com/cloudhsm/latest/userguide/java-lib-supported.html

Google Cloud Services. (3 de Nov de 2021). *Cloud Key Management Service - Key purposes and algorithms*. Obtenido de Google Cloud Services: https://cloud.google.com/kms/docs/algorithms

PrimeKey Solutions AB. (11 de December de 2021). *EJBCA product documentation*. Obtenido de EJBCA by PrimeKey: https://www.ejbca.org/documentation/

PrimeKey Solutions AB. (11 de Dec de 2021). *EJBCA ACME*. Obtenido de PrimeKey Documentation: https://doc.primekey.com/ejbca743/ejbca-operations/ejbca-ca-concept-guide/protocols/acme

PrimeKey Solutions AB. (11 de Dec de 2021). *EJBCA integration with AWS CloudHSM*. Obtenido de PrimeKey EJBCA Documentation: https://doc.primekey.com/ejbca743/ejbca-integration/hardware-security-modules-hsm/aws-cloudhsm

PrimeKey Solutions AB. (18 de December de 2021). *EJBCA REST Interface*. Obtenido de PrimeKey EJBCA Documentation: https://download.primekey.com/docs/EJBCA-Enterprise/7_2_0/EJBCA_REST_Interface.html

EJBCA REST Interface. (11 de May de 2012). *Enterprise EJBCA features vs Community*. Obtenido de PrimeKey EJBCA Blog: https://blog.ejbca.org/2012/05/new-features-in-ejbca-5.html

PrimeKey Solutions AB. (13 de Nov de 2021). *EJBCA Integrations with HSM*. Obtenido de PrimeKey EJBCA Documentation: https://doc.primekey.com/ejbca743/ejbca-integration/hardware-security-modules-hsm/generic-pkcs-11-provider

Amazon Web Services. (11 de December de 2021). *Install and Configure the AWS CloudHSM Client*. Obtenido de Amazon Web Services Documentation: https://docs.aws.amazon.com/cloudhsm/latest/userguide/install-and-configure-client-linux.html

Cryptosense. (1 de Nov de 2021). *Cloud KMS Comparision*. Obtenido de Cryptosense Blog: https://cryptosense.com/knowledge-base/cloud-cryptography-comparison

Amazon Web Services. (6 de Dec de 2021). *Getting started with AWS App Mesh and Amazon EKS*. Obtenido de Amazon Web Services Blogs: https://aws.amazon.com/blogs/containers/getting-started-with-app-mesh-and-eks/

Amazon Web Services. (6 de Dec de 2021). *Securing Kubernetes applications with AWS App Mesh and cert-manager*. Obtenido de Amazon Web Services Blogs: https://aws.amazon.com/blogs/containers/securing-kubernetes-applications-with-aws-app-mesh-and-cert-manager/

PrimeKey Solutions AB. (6 de Dec de 2021). *Issuing certificates to Kubernetes services using an external EJBCA EE*. Obtenido de PrimeKey GitHub: https://github.com/primekeydevs/containers/tree/master/deployment-examples/kubernetes/cert-manager