Implementación de Arquitectura Zero Trust en AWS (Free Tier)

Por: Cristian Jiménez

Estudiante de Ciberseguridad y Certificado en AWS Solutions Architect, AWS Cloud Security Foundations.

Introducción

En este informe se detalla el montaje de una arquitectura cloud bajo el modelo de **Zero Trust** en Amazon Web Services (AWS), empleando exclusivamente servicios disponibles en la capa **Free Tier**. El proyecto busca demostrar cómo fortalecer la seguridad en la nube aplicando los principios de Zero Trust definidos por **NIST SP 800-207**[1][2], sin recurrir a servicios de pago avanzados (como *GuardDuty* o *Security Hub*). En esencia, Zero Trust promueve el *"no confiar implícitamente en nada dentro o fuera de la red"*, exigiendo verificación continua de cada acceso y el principio de privilegios mínimos. Siguiendo este enfoque, la implementación incluye controles estrictos de identidad y acceso, micro-segmentación de la red, cifrado de datos en reposo y monitoreo continuo de actividades, todo ello aprovechando servicios estándar de AWS.

A continuación, se presenta la arquitectura general del entorno construido, indicando los principales componentes y su rol en el modelo Zero Trust:

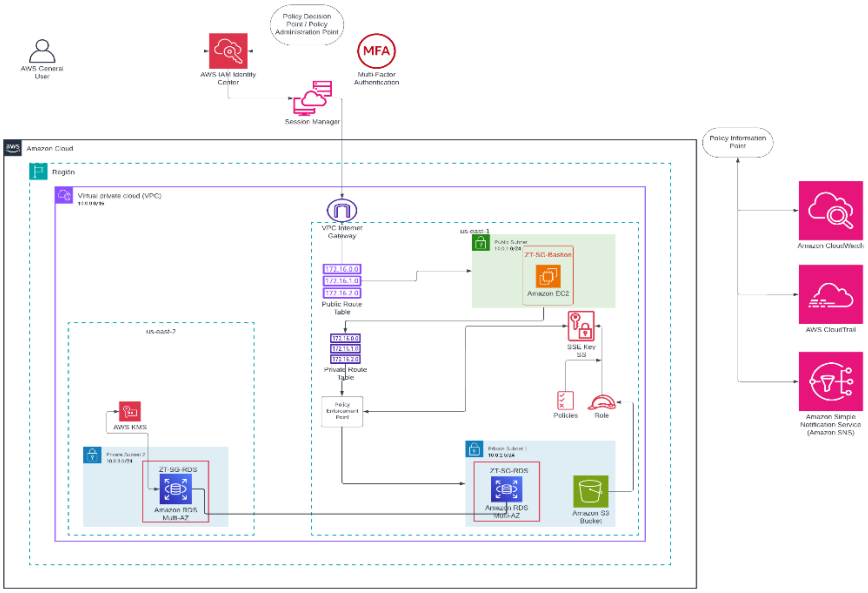

Figura: Arquitectura general del entorno Zero Trust en AWS (Free Tier).

Como se observa, el usuario administrador se autentica mediante **IAM** con **MFA**, actuando IAM Identity Center como punto de administración y decisión de políticas (**PAP/PDP**). Los recursos se aíslan dentro de una **VPC** con subred pública (para un *bastion host* de acceso) y subred privada (para una base de datos **RDS** cifrada y sin acceso público). Un bucket **S3** privado con cifrado **KMS** aloja datos protegidos. Finalmente, servicios de registro y monitoreo (**CloudTrail**, **CloudWatch**) junto con notificaciones **SNS** proveen un punto de información de políticas (**PIP**) para detección de eventos de seguridad.

En los apartados siguientes se describen los objetivos del proyecto, los conceptos clave involucrados, el desarrollo paso a paso dividido en fases (Fase 0 a Fase 7) y las evidencias recopiladas. Se concluye con una discusión sobre posibles mejoras, riesgos y conclusiones generales de la implementación.

Objetivos del Proyecto

Objetivo general: Implementar una arquitectura segura en AWS aplicando los principios de Zero Trust (ZTA) según el marco NIST SP 800-207, utilizando únicamente servicios elegibles en el Free Tier. Esto implica construir un entorno resiliente con controles de acceso estrictos, segmentación granular, cifrado de datos y monitoreo continuo, demostrando que es posible alcanzar un alto nivel de seguridad sin depender de servicios de seguridad avanzados de pago.

Objetivos específicos:

- **Seguridad de identidades:** Fortalecer la gestión de identidades habilitando autenticación multi-factor (MFA) y eliminando credenciales permanentes. Implementar un centro de identidad centralizado (**AWS IAM Identity Center**) como fuente de apoyo para autenticación/autorización, siguiendo el principio *"Nunca confiar, siempre verificar"* en cada inicio de sesión.
- **Mínimos privilegios:** Definir políticas de acceso IAM personalizadas y roles de usuario con privilegios mínimos necesarios, incorporando *condiciones* (ej. exigir conexiones cifradas) para reforzar la postura Zero Trust incluso en accesos permitidos.
- **Protección de datos sensibles:** Asegurar que los datos almacenados (p. ej. en **Amazon S3** y **Amazon RDS**) estén cifrados en reposo mediante **AWS KMS** u otros mecanismos, y que su acceso esté estrictamente controlado (sin acceso público por defecto, solo accesible a entidades autorizadas).
- **Micro-segmentación de red:** Diseñar la red cloud de forma segmentada con **VPC** privadas, subredes públicas y privadas, listas de control de acceso de red (**Security Groups**) y sin confiar en la ubicación de red. Garantizar que recursos críticos (bases de datos) no sean accesibles desde Internet, estableciendo puntos de acceso intermedios seguros (bastion) para la administración.

- **Monitoreo y trazabilidad:** Habilitar el **registro continuo de actividades** en la cuenta AWS mediante **AWS CloudTrail**, enviando estos registros a almacenamiento seguro y a un servicio de monitoreo (**CloudWatch Logs**). Configurar *alarmas* y notificaciones **SNS** para eventos críticos (p. ej. intentos de inicio de sesión fallidos), facilitando una respuesta oportuna.
- **Comprobación y evidencia de cumplimiento:** Verificar en cada fase que las medidas configuradas funcionan según lo esperado. Generar evidencias tangibles (reportes de IAM, listados de políticas, registros de CloudTrail, correos de alerta de SNS, etc.) que demuestren el cumplimiento de los principios Zero Trust en el entorno.

Conceptos Clave

A continuación, se explican brevemente los conceptos y servicios fundamentales empleados en este proyecto y su relación con el enfoque Zero Trust:

- **Zero Trust Architecture (ZTA):** Es un modelo de seguridad que **elimina la confianza implícita** en la red interna. Bajo Zero Trust, ningún usuario o dispositivo se considera confiable por defecto; toda solicitud de acceso debe autenticarse y autorizarse rigurosamente, independientemente de su origen. NIST define Zero Trust como un conjunto de paradigmas que mueve las defensas del perímetro de red tradicional hacia una protección centrada en los **recursos**, asumiendo que no se concede confianza automática basada en la ubicación de la red o la propiedad del dispositivo[1][2]. Los principios básicos incluyen *verificación continua* de identidad (por ejemplo, mediante MFA), *principio de mínimo privilegio* (otorgar solo los accesos estrictamente necesarios) y *microsegmentación* (aislar recursos para limitar el movimiento lateral).

  En una arquitectura Zero Trust se introducen roles lógicos: el **PAP** (*Policy Administration Point*) administra las políticas de seguridad, el **PDP** (*Policy Decision Point*) toma las decisiones de acceso evaluando esas políticas frente a cada petición, el **PEP** (*Policy Enforcement Point*) es el punto que aplica o ejecuta la decisión (permitiendo o negando el acceso al recurso), y el **PIP** (*Policy Information Point*) provee información contextual (identidad, estado del dispositivo, logs, etc.) para apoyar las decisiones y auditorías. Todo el diseño realizado sigue estos principios: autenticación estricta antes de otorgar acceso, control de accesos en cada capa y monitoreo/telemetría constante de las actividades.

- **AWS Identity and Access Management (IAM):** IAM es el servicio central de AWS para la gestión de identidades (usuarios, grupos, roles) y sus permisos de acceso. Permite controlar de forma segura quién está autenticado y autorizado a usar cada recurso de AWS[3]. En este proyecto se usa IAM no para usuarios convencionales (se evita crear usuarios IAM de largo plazo), sino para definir **roles IAM** (identidades temporales asignables) y **políticas IAM** personalizadas que implementan el principio de privilegio mínimo. Las políticas de IAM son documentos JSON que especifican qué acciones están permitidas o denegadas sobre qué recursos, pudiendo incluir condiciones (ej. requerir que la conexión sea vía TLS). Los **roles IAM** son identidades que pueden ser asumidas temporalmente por usuarios u otras entidades autorizadas, obteniendo credenciales temporales; esto reduce la necesidad de credenciales permanentes, alineado con Zero Trust (sesiones de acceso de corta duración y *just-in-time*). Adicionalmente, AWS IAM proporciona la infraestructura necesaria para manejar autenticación y autorización en AWS[4], funcionando en nuestro diseño como **PAP/PDP** (donde residen y se evalúan las políticas de acceso) y en combinación con otros servicios para aplicar esas decisiones.

- **AWS IAM Identity Center (AWS SSO):** Es un servicio de AWS (anteriormente llamado AWS SSO) que permite la gestión centralizada de identidades corporativas y el inicio de sesión federado en la Consola AWS con credenciales temporales. En este proyecto, Identity Center actúa como **portal de autenticación** con soporte de MFA para los administradores, evitando el uso directo del usuario raíz de la cuenta o de usuarios IAM locales. El Identity Center nos permite configurar usuarios administradores con MFA, *permission sets* (conjuntos de permisos predefinidos como Administrador, OnlyRead, etc.) e integrar con IAM roles. En la arquitectura Zero Trust, Identity Center asume el rol de **PAP** (Policy Administration Point a nivel identidad) y **PDP** (Policy Decision Point, validando credenciales del usuario y otorgando tokens de sesión con los permisos adecuados) antes de que un usuario pueda acceder a recursos AWS. Esto mejora la seguridad al no tener que crear usuarios IAM permanentes y forzar MFA en cada inicio de sesión (*"verificación continua"*).

- **Multi-Factor Authentication (MFA):** Es un mecanismo de autenticación que requiere al usuario proporcionar más de un factor (por ejemplo, algo que sabe -contraseña- **y** algo que tiene -token MFA-). Habilitar MFA en la cuenta raíz de AWS y para los usuarios de Identity Center añade una capa fundamental de seguridad: incluso si una contraseña se ve comprometida, el atacante no podría acceder sin el segundo factor. MFA implementa el principio de Zero Trust de *"nunca confiar, siempre verificar"*, ya que cada inicio de sesión debe confirmarse con un factor adicional. En nuestra arquitectura, MFA está activado para todos los accesos humanos: el usuario administrador recibe un código temporal en su app de autenticación al iniciar sesión, lo que actúa como **PIP** (fuente de información de autenticación contextual) alimentando al Policy Decision Point. En definitiva, MFA reduce drásticamente el riesgo de accesos no autorizados con credenciales robadas.

- **Amazon Virtual Private Cloud (VPC):** Servicio de AWS que permite aprovisionar una red virtual aislada dentro de la nube AWS. Una **VPC** define un espacio de direcciones IP privadas en el cual podemos lanzar recursos (EC2, RDS, etc.) y controlar su conectividad. En este proyecto se creó una VPC dedicada (*10.0.0.0/16*) subdividida en **subredes**: una subred pública (*10.0.1.0/24*, con acceso a Internet) para hospedar un *bastion host*, y una o más subredes privadas (*10.0.2.0/24, 10.0.3.0/24*, sin acceso directo a Internet) para la base de datos y otros recursos internos. La VPC está asociada a un **Internet Gateway** (IGW) pero **sólo** la subred pública tiene ruta hacia él; la subred privada no tiene rutas de salida a Internet. Esto implementa *microsegmentación*: la red interna se divide en zonas de distinta confianza, minimizando la superficie expuesta. Dentro de la VPC se usan **Security Groups** (firewalls virtuales por instancia/servicio) para restringir aún más el tráfico: por ejemplo, el SG del bastion solo permite SSH desde la IP del administrador, y el SG de la base de datos solo acepta conexiones desde el SG del bastion (no desde cualquier IP). La VPC y sus componentes actúan como **Policy Enforcement Point** en el modelo Zero Trust, al aislar y controlar el flujo de red: ningún recurso interno confía en la red por defecto, solo se abren las comunicaciones explícitamente permitidas.

- **Amazon EC2 (Bastion Host):** Amazon Elastic Compute Cloud (EC2) provee máquinas virtuales en la nube. Un **Bastion Host** es un servidor fortificado que sirve como punto de entrada seguro a una red privada. En esta arquitectura, el bastion (instancia EC2 Amazon Linux 2 t2.micro) reside en la subred pública y es la **única** máquina con acceso SSH desde el exterior (y aun así limitado a la IP del administrador). Su propósito es ser la puerta de enlace para administrar los recursos en la subred privada (por ejemplo, conectarse a la base de datos RDS), evitando exponer directamente esos recursos al exterior.

Para reforzar la filosofía Zero Trust, se habilitó **AWS Systems Manager Session Manager** en el bastion, de forma que incluso la conexión a éste pueda realizarse **sin abrir puertos SSH** ni manejar claves privadas. El bastion host, con sus controles de acceso, actúa como **PEP** (Policy Enforcement Point) ya que todos los accesos a la red interna pasan por él y solo se permiten desde orígenes confiables. En caso de compromiso de la instancia bastion, los recursos privados aún están protegidos por credenciales y cifrado, pero igualmente se toman medidas para asegurar el bastion (acceso restringido, parches, monitoreo vía SSM).

- **Session Manager:** Es una característica de AWS Systems Manager que permite establecer **sesiones remotas seguras** hacia instancias EC2 sin necesidad de abrir puertos ni usar SSH/RDP. Session Manager abre una consola/terminal dentro de la instancia mediante un túnel TLS administrado por AWS, autenticando al usuario vía IAM y registrando toda la actividad de la sesión. Con Session Manager, el bastion host pudo administrarse sin habilitar puertos de entrada (22) en absoluto, aumentando la seguridad de la arquitectura. **Session Manager es un servicio totalmente administrado** que proporciona administración segura de nodos sin necesidad de hosts bastión tradicionales ni claves SSH. Implementa controles de acceso centralizados mediante políticas IAM (solo usuarios con permisos SSM pueden iniciar sesiones). Además, genera registros detallados de las conexiones y comandos ejecutados, integrándose con CloudWatch Logs o S3 para auditoría. En el contexto Zero Trust, Session Manager actúa como **PEP** al establecer un canal de administración fuertemente autenticado y autorizado para acceder a instancias, y también como **PIP** al proveer registros de actividad de cada sesión (quién accedió, cuándo y qué hizo) para fines de monitoreo. El uso de Session Manager en lugar de SSH cumple con *"nunca confiar en la red"*, ya que no se exponen puertos ni se asume que una IP interna es segura; todo acceso se verifica mediante IAM y se registra.

- **Amazon Simple Storage Service (S3):** Servicio de almacenamiento de objetos escalable de AWS. S3 es utilizado en este proyecto para dos fines: *(a)* crear un **bucket seguro** para almacenar archivos privados de forma cifrada y con políticas de acceso Zero Trust, y *(b)* almacenar registros de auditoría (logs de CloudTrail). El **bucket de datos** ("zta-secure-bucket") se configuró con *Block Public Access* habilitado (todas las opciones en "block public access" activadas) para garantizar que ningún objeto pueda hacerse público accidentalmente. Asimismo, se habilitó el **versionado** de objetos y el **cifrado por defecto** (Server-Side Encryption) con claves administradas de S3 (SSE-S3, AES-256), de modo que cualquier archivo subido se almacena cifrado automáticamente. Para reforzar las políticas Zero Trust en S3, se implementaron dos capas de control: una **política de IAM** a nivel de usuario/rol que solo permite operaciones en el bucket si la conexión es segura (HTTPS), y una **política de bucket S3** a nivel de recurso que **deniega** cualquier

carga (PUT) que no esté cifrada o que provenga de conexión no cifrada. Estas políticas aseguran que **solo conexiones TLS** pueden leer/listar objetos y que **solo objetos cifrados** pueden incorporarse al bucket (si alguien intentara subir algo sin cifrado del lado servidor, se rechaza). En conjunto, S3 con cifrado y políticas estrictas se convierte en un repositorio confiable para datos sensibles, alineado con Zero Trust (sin acceso implícito desde Internet, datos ilegibles sin claves, y verificación de requisitos de seguridad en cada petición). S3 también aloja los logs de CloudTrail; ese bucket de logs se configura igualmente privado y cifrado para que solo roles internos de auditoría puedan leerlos. En la arquitectura Zero Trust, el bucket S3 seguro actúa como **recurso protegido** cuya protección se implementa mediante PEP (políticas de bucket que deniegan accesos no confiables) y PAP/PDP (políticas IAM que definen condiciones para acceder).

- **AWS Key Management Service (KMS):** Servicio administrado de gestión de claves criptográficas. KMS es fundamental para habilitar cifrado en varios niveles de la arquitectura: se utiliza la **clave KMS por defecto de AWS (aws/s3)** para cifrar objetos en S3, la clave **aws/rds** para cifrar los datos de la base de datos RDS, y también se podría usar una KMS CMK (Customer Master Key) propia si quisiéramos controlar las políticas de cifrado en detalle. En este laboratorio, se optó por las claves administradas por AWS para simplicidad, pero **toda la data en reposo está cifrada**. KMS asegura que solo las entidades con permisos adecuados puedan descifrar los datos. Por ejemplo, al activar cifrado en RDS, los datos en disco y respaldos se cifran con la clave *aws/rds*. En Zero Trust, el cifrado robusto es esencial para la **confidencialidad** de los recursos; KMS proporciona el *PAP* para administrar las políticas de uso de claves (quién puede desencriptar) y el *PEP* al realizar la operación de cifrado/descifrado bajo control de dichas políticas. Aunque KMS no decide accesos al recurso en sí, sí decide sobre el acceso a las claves de cifrado – un atacante que consiguiera volcar una base de datos cifrada no podría leerla sin acceso KMS autorizado. En otras palabras, KMS refuerza la capa de protección de datos: incluso si otras medidas fallaran, los datos permanecerían inaccesibles sin las credenciales apropiadas de KMS.

- **Amazon Relational Database Service (RDS):** Servicio de base de datos administrada. En la implementación se desplegó una instancia **RDS MySQL** de nivel gratuito, configurada **sin acceso público** (flag *Public access: No*) para que **solo esté accesible dentro de la VPC privada**. La instancia utiliza el **grupo de subredes** privado creado (que abarca dos subredes privadas en distintas AZ para alta disponibilidad), garantizando que no tenga ni IP pública ni ruta de salida a Internet. El único modo de conectarse a esta base de datos es desde la instancia bastion dentro de la misma VPC. Además, el RDS está **cifrado con KMS** (se habilitó “Enable encryption” al crear la BD, usando la clave por defecto aws/rds), de forma que los datos en reposo, *snapshots* y

respaldos estén cifrados de manera transparente. Para controlar el acceso en la red, el RDS usa un **Security Group** (ZT-SG-RDS) que solo permite tráfico entrante desde el SG del bastion en el puerto de la base de datos (3306 para MySQL). No se utilizan credenciales IAM directas para el acceso a la BD en este caso (se usó usuario admin propio de MySQL con contraseña segura), pero como capa adicional de trazabilidad se habilitaron los **logs de auditoría** de RDS hacia CloudWatch (ejecutando algunas consultas para generar eventos). Desde la perspectiva de Zero Trust, la instancia RDS es un **recurso altamente protegido** ya que está aislada en una red de confianza mínima (solo el bastion puede verla), sus datos están cifrados, y todas las operaciones quedan registradas. No confía en la red (no IP pública) ni en otros servicios por defecto. Su protección se asegura mediante PEPs de red (security groups, subred privada) y controles PAP/PDP (configuraciones que impiden accesos no autenticados y requieren cifrado).

- **AWS CloudTrail:** Es el servicio de AWS para **auditoría y registro de actividades** en la cuenta. CloudTrail registra todos los eventos de la Consola, llamadas a APIs, acciones realizadas por usuarios IAM/SSO, cambios en configuraciones, etc., y los entrega en forma de archivos de log. En este proyecto, CloudTrail fue activado a nivel de cuenta y región (*Trail* llamado "ZT-CloudTrail") para capturar tanto **eventos de administración** (ej. creación de usuarios, cambios de seguridad) como **eventos de datos** en S3 (accesos al bucket). Se configuró para enviar los logs a un bucket S3 dedicado ("zt-cloudtrail-logs") con cifrado SSE habilitado para esos archivos. Adicionalmente, se integró CloudTrail con CloudWatch Logs para análisis en tiempo real (ver siguiente punto). El rol de CloudTrail en Zero Trust es proporcionar **visibilidad completa** de lo que ocurre en el entorno – es un claro **Policy Information Point (PIP)**, reuniendo información que permite detectar violaciones de políticas o comportamientos anómalos. Por ejemplo, CloudTrail nos permitirá ver si alguien intenta usar credenciales incorrectas, si se crean nuevas claves de acceso, o si se cambia una política de forma inesperada. Sin este rastro auditado, sería difícil aplicar el principio de *"asume la brecha"* (assume breach) y responder a incidentes. AWS CloudTrail es esencial para la gobernanza, cumplimiento y análisis forense en AWS. En nuestro caso, habilitar CloudTrail asegura que cada acción relevante queda registrada y almacenada de forma segura (en un bucket cifrado que solo admins pueden leer). Esto soporta la capacidad de **verificación continua** de Zero Trust – las políticas pueden ser ajustadas o aplicadas dinámicamente en base a estos registros, y ninguna acción pasa inadvertida.

- **Amazon CloudWatch (Logs, Metrics y Alarms):** Amazon CloudWatch es el servicio de **monitorización y observabilidad** de AWS. Cumple varios roles: centraliza logs de servicios, recoge métricas de rendimiento, y permite crear **alarmas** basadas en condiciones definidas. En esta arquitectura, CloudWatch se usa principalmente para: *(a)* recibir los **logs de CloudTrail** en un *Log Group* (llamado "zt-cloudtrail-logs") para poder buscar patrones de eventos, y *(b)* definir una **alarma** de seguridad que detecte intentos de inicio de sesión fallidos en la Consola AWS. Los **Logs de CloudWatch** actúan como repositorio en tiempo real donde aplicamos filtros: por ejemplo, se creó un *Metric Filter* en el log group de CloudTrail para buscar eventos de tipo `ConsoleLogin` fallidos (con `errorMessage "Failed authentication"`). Este filtro contabiliza cada intento fallido y genera una métrica personalizada "FailedConsoleLogins". Sobre esa métrica, definimos una **alarma** que se dispare cuando su valor sea ≥ 1 en un período de 5 minutos, es decir, si ocurre al menos un login fallido. Al activarse la alarma, configuramos que envíe una notificación mediante **Amazon SNS** (un tema llamado "security-alerts"). CloudWatch por tanto está realizando tanto la función de detección (mirando patrones en los logs) como la de notificación (a través de la alarma). En terminología Zero Trust, CloudWatch funciona como **PIP** (brinda información sobre eventos en forma de métricas y logs) y también como un "**mini-PDP"** automatizado para ciertas condiciones (decide que una situación –varios login fallidos– amerita una alerta). Su integración con SNS nos permite tener *telemetría accionable*: no solo se registran los eventos, sino que inmediatamente se genera una respuesta (un correo de alerta al administrador) ante indicadores de potencial incidente. Cabe destacar que CloudWatch también se puede usar para muchas otras métricas (uso de CPU, memoria, etc.), pero en el contexto de seguridad nos centramos en eventos críticos. Este monitoreo continuo asegurado por CloudWatch satisface la recomendación de NIST de tener *"visibilidad y análisis constantes"* en arquitecturas Zero Trust.

- **Amazon Simple Notification Service (SNS):** Servicio de mensajería *pub/sub* totalmente gestionado que permite enviar notificaciones a múltiples suscriptores (vía email, SMS, etc.) de forma automática. En la solución, SNS se utiliza para **enviar alertas de seguridad** al equipo administrador. Se creó un tópico SNS ("security-alerts") al que se suscribió la dirección de correo del administrador de seguridad; CloudWatch Alarm está configurado para publicar un mensaje en este tópico cuando detecta un evento de interés. En ese momento, SNS envía un email al administrador con el contenido de la alerta (indicando, por ejemplo, que hubo un *Failed Console Login* a cierta hora). SNS garantiza una entrega oportuna y confiable de las notificaciones. Aunque SNS no interviene directamente en la autorización de accesos, sí forma parte de la **respuesta** ante incidentes, facilitando el principio de Zero Trust de *"asegurar, monitorear y responder"*.

Gracias a SNS, la arquitectura no solo recopila información (logs) sino que **actúa** sobre ella notificando a personas o sistemas para que tomen medidas. En cierto modo, SNS complementa al PIP/PDP al cerrar el ciclo de seguridad: es un canal de comunicación que asegura que las decisiones (ej. generar una alarma) se traduzcan en acciones (un ser humano informado, o potencialmente un *lambda function* que remedie el problema). Amazon SNS es un servicio de mensajería desacoplada; en nuestro caso lo utilizamos en modo *Application-to-Person*, enviando email, pero también podría integrarse con flujos automatizados más avanzados (llamar a Lambda, crear tickets, etc.).

Desarrollo de la Solución por Fases

A continuación, se presenta el desarrollo del proyecto dividido en **8 fases** (desde la Fase 0 hasta la Fase 7). Cada fase corresponde a un conjunto lógico de configuraciones implementadas en el entorno AWS, siguiendo un orden secuencial. Para cada fase se explica su objetivo, las acciones realizadas paso a paso y cómo contribuye al modelo Zero Trust. Al final de cada fase, se incluye una tabla resumen con los **servicios/componentes configurados**, su **propósito** dentro de la arquitectura, y su **rol** en términos de Zero Trust (PAP, PDP, PEP, PIP, etc., según corresponda).

Fase 0: Configuración Inicial de Identidades Seguras (Root MFA e Identity Center)

Objetivo: Establecer una base sólida de seguridad de identidad en la cuenta AWS antes de cualquier otro despliegue. Esto incluye proteger la cuenta **root** con MFA y habilitar un **centro de identidades** (AWS IAM Identity Center) para gestionar accesos administrativos sin usar las credenciales raíz en operaciones diarias. Con esta fase se busca implementar los puntos de **Policy Administration Point (PAP)** y **Policy Decision Point (PDP)** iniciales de la arquitectura Zero Trust, garantizando que *solo* identidades autenticadas de forma robusta puedan proceder a operar en AWS.

Pasos realizados:

1. **Habilitar MFA en la cuenta root:** Se ingresó a la consola de AWS con el usuario raíz de la cuenta (creado al iniciar el Free Tier) y se configuró MFA (Multi-Factor Authentication) en la sección *Security Credentials*. Se utilizó una aplicación de autenticación (Microsoft Authenticator) para escanear el código QR proporcionado y asociar un dispositivo MFA al usuario root. A partir de entonces, cualquier login como root requiere no solo la contraseña sino también el código dinámico de 6 dígitos de la app MFA, fortaleciendo la protección del superusuario.

2. **Configurar AWS IAM Identity Center (AWS SSO):** Desde la consola se activó **AWS IAM Identity Center** (servicio que unifica identidades para la organización, incluso en un solo entorno de cuenta).

En Identity Center:

3. Se creó un **usuario administrativo** separado (p. ej. "ICDD") proporcionando un nombre de usuario y email (se usó un correo personal). Se dejó que Identity Center enviara automáticamente un email al usuario para establecer su contraseña. Opcionalmente, se pudo agrupar en un grupo "ZT-Admins", aunque no fue estrictamente necesario en este entorno.
4. Al usuario "ICDD" se le asignó un **Permission Set** predefinido de *AdministratorAccess*, otorgándole efectivamente permisos de administrador dentro de la cuenta AWS, pero a través del SSO. Esto significa que "ICDD" no tiene una clave de acceso permanente, sino que obtiene credenciales temporales cuando inicia una sesión federada.
5. El usuario recibió vía correo un enlace para activar su cuenta SSO. Tras establecer su contraseña e iniciar sesión en el *User Portal* de AWS SSO, también se le forzó a registrar un dispositivo MFA para su usuario SSO (Identity Center soporta MFA para cada usuario gestionado). Se eligió usar una aplicación móvil de autenticación como segundo factor. De este modo, **todos los accesos interactivos al entorno AWS exigirán MFA**, alineándose con la verificación continua de Zero Trust.
6. En la configuración de AWS IAM Identity Center → *Settings*, se habilitó la opción "Require MFA **every time** they sign in" (MFA siempre), de forma que incluso si el usuario cierra y vuelve a abrir la sesión del portal SSO, se le pedirá MFA cada vez.

7. **Probar inicio de sesión SSO con MFA:** Se verificó que el nuevo usuario puede iniciar sesión en AWS a través del portal SSO. Al navegar al URL personalizado de Identity Center (formato `https://d-xxxx.awsapps.com/start`), el usuario ingresa su email y contraseña, y luego el código MFA de su dispositivo móvil. Una vez autenticado, aparece en el portal el icono de la cuenta AWS asignada con *Administrator Access*; al hacer clic, se inicia la sesión federada en la Consola AWS con privilegios de administrador. Se confirmó que **no se usaron en ningún momento credenciales estáticas de IAM** – el acceso ocurrió vía SSO con MFA. En este punto, la cuenta root queda reservada solo para emergencias o tareas especiales, y la operación cotidiana se hará con el usuario federado IC-Admin.

Al finalizar esta fase, **todas las identidades utilizadas para administrar la nube están fuertemente autenticadas** y administradas centralmente. Se logra así que la superficie de ataque de identidad se reduzca (no hay usuarios IAM permanentes ni claves sin MFA).

La siguiente tabla resume los componentes configurados en esta fase y su aporte al modelo Zero Trust:

Servicio/Elemento	Propósito en la arquitectura Zero Trust	Rol ZT
IAM Identity Center (AWS SSO)	Gestionar identidades centralmente, federación con MFA obligatoria. Facilita administración unificada de acceso.	**PAP / PDP** (administra políticas de identidad y decide autenticación)
MFA habilitado (usuario root & SSO)	Aportar verificación adicional en cada inicio de sesión, evitando accesos solo con contraseña. Refuerza “confianza continua”.	**PIP** (factor de información de autenticación contextual para decisiones)
Credenciales temporales (SSO roles)	Eliminar usuarios IAM de largo plazo; todo acceso ocurre mediante asunciones de rol temporales. Reduce riesgo de credenciales comprometidas permanentes.	**PEP** (aplicación de políticas de sesión por rol; no hay acceso sin pasar por la federación)

Resultado: Tras la Fase 0, el administrador inicia sesión mediante AWS SSO con MFA y accede a la consola AWS sin usar credenciales estáticas. Se ha establecido el **punto de entrada seguro** al entorno AWS: solo el usuario federado con MFA (PDP/PAP) puede administrar recursos, cumpliendo el principio Zero Trust de “nunca confiar en la identidad por defecto, verificar siempre” (cada sesión requiere MFA). Esto sienta las bases para las siguientes fases donde se crearán los recursos cloud sabiendo que cualquier acción estará autenticada y auditada.

Fase 1: Auditoría de Identidades y Control de Acceso con Mínimo Privilegio (IAM)

Objetivo: Consolidar la gestión de identidades aplicando el **principio de mínimo privilegio** y habilitando capacidades de **auditoría** sobre las configuraciones IAM. Aunque ahora los accesos se realizan vía Identity Center, AWS IAM sigue desempeñando un rol crítico en la definición de políticas detalladas. En esta fase se crean políticas y roles IAM personalizados para restringir accesos (ej. a S3) bajo ciertas condiciones, y se utilizan las herramientas de AWS para auditar el estado de las identidades/credenciales. Esta fase refuerza los componentes de **PAP/PDP** de Zero Trust, asegurando que incluso usuarios autenticados tengan solo los permisos necesarios y que exista visibilidad sobre cualquier credencial existente.

Pasos realizados:

1. **Revisar asignaciones en Identity Center:** Primero se corroboró en AWS IAM Identity Center que el usuario SSO "ICDD" tiene efectivamente el permiso **AdministratorAccess** asignado a la cuenta AWS. Adicionalmente, para simulaciones futuras, se creó un **Permission Set** secundario llamado "ZTA-ReadOnly" con la política *ReadOnlyAccess* de AWS. Esto permite que el mismo usuario pueda alternar a un rol de solo lectura en caso de ser necesario (*simulando un contexto de menor privilegio*). Tener disponibles múltiples perfiles de permisos para un usuario es útil en Zero Trust para aplicar privilegios mínimos según contexto (por ejemplo, usar el rol de lectura para tareas que no requieran escribir cambios). Se verificó que en la consola SSO el usuario ahora ve dos opciones de inicio de sesión: *Admin Access* y *ReadOnly*.

2. **Crear política IAM condicional (ejemplo S3):** Usando la consola de **IAM**, se definió una política personalizada para restringir acceso a un bucket S3 solo si la conexión es segura. Desde *IAM* > *Policies* > *Create Policy*, en modo JSON se introdujo la siguiente política (nombre: `ZT-Restricted-S3-Access`):

```
{
  "Version": "2012-10-17",
  "Statement": [{
    "Sid": "AllowReadAccessToSpecificBucket",
    "Effect": "Allow",
    "Action": [
      "s3:GetObject",
      "s3:ListBucket"
    ],
    "Resource": [
      "arn:aws:s3:::zt-secure-bucket",
      "arn:aws:s3:::zt-secure-bucket/*"
    ],
    "Condition": {
```

```
      "Bool": { "aws:SecureTransport": "true" }
    }
  }]
}
```

Esta política permite listar y leer objetos del bucket `zt-secure-bucket` **solo** si la conexión se realiza por HTTPS (`aws:SecureTransport` debe ser `"true"`). Cualquier intento de acceder sin cifrado (HTTP) sería denegado automáticamente por esta condición. Este tipo de restricción implementa Zero Trust a nivel de servicio: incluso un usuario autenticado no puede extraer datos de S3 por un canal no seguro. Se guardó la política y IAM la mostró en la lista de políticas custom. El **resultado esperado** es tener una política IAM que aplica acceso condicional (requiere transporte cifrado), alineada con Zero Trust.

3. **Crear un rol IAM con mínimos privilegios:** A continuación, se creó un **Role IAM** para aprovechar la política recién definida. En *IAM* > *Roles* > *Create Role*:

4. Tipo de entidad confiable: **AWS account** (esta cuenta actual). Esto define que el rol podrá ser asumido por identidades de la misma cuenta (p. ej. nuestro usuario SSO a través de un Permission Set).
5. Se adjuntó la política **ZT-Restricted-S3-Access** al rol. No se añadieron otras políticas, por lo que este rol tendrá permisos muy limitados (solo lectura de ese bucket y únicamente vía TLS).
6. Se asignó nombre al rol, por ejemplo `ZT-S3-Auditor`.
   Una vez creado, el rol *ZT-S3-Auditor* aparece en la lista de roles de IAM.

7. **Integrar el rol con Identity Center:** Para permitir que nuestro usuario federado asuma este nuevo rol de forma fácil, volvimos a AWS IAM Identity Center > AWS Accounts > Assignments. Seleccionamos la cuenta, elegimos el usuario “ICDD”, y asignamos un **nuevo Permission Set** que mapea directamente al rol IAM `ZT-S3-Auditor`. Esto se hace mediante la opción de *Custom Permission Set*, especificando el ARN del rol IAM. Después de este paso, cuando “ICDD” inicia sesión por SSO, verá una tercera opción de inicio: la cuenta AWS con el rol `ZT-S3-Auditor`. Con un clic podrá asumir ese rol de auditoría de S3.

Resultado: ahora el usuario SSO puede alternar entre *Admin*, *ReadOnly* y *S3-Auditor* según la tarea requerida. Esto ejemplifica la separación de privilegios por contexto (principio Zero Trust: otorgar solo los accesos necesarios en cada caso, evitando sobreprivilegios). Por ejemplo, para revisar contenido del bucket seguro podría usar el rol de auditoría sin arriesgar comandos de más alcance.

8. **Auditar usuarios, roles y políticas actuales:** Con la configuración de identidad en su lugar, se procedió a ejecutar una serie de comandos de **AWS CLI** (usando AWS CloudShell) para auditar el estado de la cuenta IAM:

```
aws iam list-users
aws iam list-roles
aws iam list-policies --scope Local
aws iam generate-credential-report
```

Estos comandos listan los **usuarios IAM** existentes, **roles IAM** existentes, **políticas IAM** locales (custom) y generan un **reporte de credenciales** respectivamente. La salida de `aws iam list-users` confirmó que **no hay usuarios IAM tradicionales** en la cuenta, lo cual es deseable porque estamos usando Identity Center en su lugar (los usuarios de SSO no aparecen en IAM). Esto demuestra que el entorno está "limpio" de identidades permanentes, reduciendo superficie de ataque. La salida de `list-roles` enumeró roles, incluyendo el rol `ZT-S3-Auditor` recién creado y roles por defecto. La salida de `list-policies --scope Local` mostró nuestras políticas personalizadas (p.ej. `ZT-Restricted-S3-Access` y el *permission set* de ReadOnly que Identity Center creó internamente). Esto es útil para verificar que solo existan las políticas esperadas y ninguna extraña. Por último, `generate-credential-report` produjo un reporte en background con el estado de todas las credenciales IAM (usuarios, últimas veces de uso, si tienen MFA, etc.). Para verlo, se usó: `aws iam get-credential-report --query 'Content' --output text | base64 --decode > iam-credential-report.csv`, lo que decodificó el CSV generado y lo guardó en un archivo local. Al inspeccionar este CSV, se verificó que el único usuario listado es "<root_account>" con MFA habilitado, y que no hay otras credenciales activas – confirmando que la única identidad persistente (root) está protegida con MFA, y que no hay cuentas huérfanas ni claves de acceso sin usar. *(Esta es una buena práctica de hardening: revisar periódicamente el credential report para detectar cuentas sin MFA, claves inactivas, etc.)*

Al concluir la Fase 1, se logró una **consolidación de la seguridad de identidades**: no existen *backdoors* de usuarios IAM sin controlar, los administradores usan roles federados con MFA, y se han introducido políticas de mínimo privilegio para casos específicos (S3). También contamos con reportes de referencia para futuras auditorías. Esta fase completa los pilares de **PAP/PDP** de Zero Trust en donde las reglas de acceso están definidas y podemos tomar decisiones informadas a la vez que empieza a introducir **PEP** (el rol ZT-S3-Auditor actúa como un Enforcement Point limitado) y mecanismos de **PIP** (reportes, CLI) para evidenciar la seguridad.

La siguiente tabla resume esta fase:

Servicio/Elemento	Propósito en la arquitectura Zero Trust	Rol ZT
AWS IAM (Políticas personalizadas)	Definir permisos granulados y condicionales (ej. exigir TLS para S3), implementando el principio de mínimo privilegio a nivel IAM.	**PAP** (gestiona políticas de acceso) *(PDP en tiempo de evaluación de condiciones)*
AWS IAM (Roles IAM)	Aplicar y delimitar los permisos otorgados a identidades federadas. El rol ZT-S3-Auditor sirve de *Policy Enforcement Point* limitado a S3 (y solo bajo ciertas condiciones).	**PEP** (contiene permisos restringidos que se aplican al ser asumido, controlando efectivamente el acceso al recurso)
AWS CLI / Credential Reports	Auditar y validar las identidades y credenciales activas en la cuenta, proporcionando visibilidad de cumplimiento (ej. confirmar MFA en root, ausencia de usuarios IAM huérfanos).	**PIP** (provee información de seguridad y cumplimiento - NIST SP 800-53 CA-7)

Resultado: Después de Fase 1, la cuenta AWS tiene identidades federadas con MFA (ningún usuario IAM directo), roles y políticas que imponen restricciones de Zero Trust (por ejemplo, acceso S3 solo por canal cifrado), y se cuenta con reportes que evidencian un estado seguro (no hay credenciales expuestas sin proteger). Esto representa la **consolidación de PAP/PDP** en el modelo Zero Trust de nuestro entorno: las políticas están establecidas y bajo control central, listas para ser aplicadas en las siguientes fases a los diferentes recursos cloud.

Fase 2: Protección de Datos en Almacenamiento (Bucket S3 Seguro con Cifrado)

Objetivo: Configurar un **almacenamiento seguro de datos** aprovechando Amazon S3, implementando controles Zero Trust a nivel de datos. En concreto, crear un **bucket S3 privado** destinado a almacenar información confidencial, garantizando que todo acceso esté autenticado, cifrado en tránsito y en reposo, sin excepciones. Se activarán políticas y características en S3 para: bloquear acceso público, exigir **cifrado de objetos (SSE)**, mantener versiones (integridad) y auditar accesos vía CloudTrail. Esta fase se enfoca en la **confidencialidad e integridad de datos**, alineando S3 como recurso con los principios Zero Trust de *no trust por defecto* y *seguridad inherente de datos* incluso si perimetrales fallan.

Pasos realizados:

1. **Crear el bucket S3 "zta-secure-bucket":** Desde la consola S3 (o usando AWS CLI), se creó un nuevo bucket de nombre único, por ejemplo `zta-secure-bucket`. Se usó la región **us-east-1** para consistencia con otros servicios. Vía CLI el comando fue:

   ```
   aws s3 mb s3://zta-secure-bucket --region us-east-1
   ```

Alternativamente en la consola S3: *Create bucket* con opciones por defecto. Importante: no se habilitó acceso público ni opciones especiales en este momento, solo crear el contenedor.

1. **Bloquear acceso público total:** Tras la creación, se configuró el bucket para **bloquear cualquier acceso público**. En la pestaña *Permissions* del bucket, sección *Block Public Access (BPA)*, se activaron **todas las opciones**: "Block *all* public access", incluyendo bloquear ACLs públicas, bloques a políticas públicas nuevas, etc.. Esto asegura que ninguna política futura ni configuración podrá exponer el bucket inadvertidamente al público. En S3, incluso si alguien tratase de otorgar acceso público, estas opciones lo anularían. *Nota:* Por defecto, S3 crea los buckets con BPA activado (desde 2021), pero es buena práctica verificarlo. Con este paso, hemos implementado un **control preventivo fuerte**: por diseño, ningún objeto de `zta-secure-bucket` será accesible sin autenticación AWS. En terminología Zero Trust, se elimina cualquier "confianza implícita" (como las que existían en modelos antiguos donde un bucket por defecto podía ser leído públicamente) – aquí, el bucket no confía en nadie externo.

2. **Habilitar Versioning y cifrado por defecto en el bucket:** En la pestaña *Properties* del bucket:

3. Se habilitó **Bucket Versioning** (cambio de *Disabled* a *Enabled*). Esto permite conservar versiones previas de objetos modificados o borrados.

El Versioning cumple un rol importante en integridad: si se altera o elimina un archivo por error o malicia, queda rastro o posibilidad de restauración.

4. Se habilitó **Default Encryption**, eligiendo "*Server-side encryption with Amazon S3 managed keys (SSE-S3)*". Esto significa que S3 cifrará automáticamente cada objeto con una clave AES-256 administrada por AWS S3 al almacenarlo (sin costo extra ni necesidad de KMS personal para este volumen). Alternativamente, podríamos usar SSE-KMS con una CMK propia, pero en la práctica SSE-S3 provee un fuerte cifrado y simplifica la configuración. Tras habilitarlo, **cada objeto que se suba al bucket será cifrado en reposo con AES-256** de manera transparente.

5. **Subir un archivo de prueba y verificar cifrado:** Para comprobar la configuración, se realizó la siguiente secuencia (usando CloudShell):

```
echo "Prueba Zero Trust S3" > prueba.txt
aws s3 cp prueba.txt s3://zta-secure-bucket/
aws s3api head-object --bucket zta-secure-bucket --key prueba.txt
```

Esto crea un archivo de texto simple, lo sube al bucket, y luego obtiene los metadatos del objeto en S3. En la salida de `head-object`, se observó el header `"ServerSideEncryption": "AES256"`, confirmando que el objeto quedó cifrado con SSE-S3. También se verificó que el objeto tenía un `VersionId` asignado debido al versioning. **Resultado esperado:** el archivo se almacena cifrado y versionado automáticamente. Esta prueba muestra que, aun si un usuario olvidara cifrar antes de subir, el bucket aplica cifrado por él. La política de Zero Trust de cifrado en reposo se cumple sin depender de la memoria del usuario.

1. **Implementar política de bucket Zero Trust:** Aunque ya el bucket está inaccesible públicamente y cifra todo, se añadió una capa extra creando una **Bucket Policy** que refuerza los requerimientos de seguridad. Se creó un archivo local `zt-s3-policy.json` con el siguiente contenido:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "DenyUnsecuredTransport",
      "Effect": "Deny",
      "Principal": "*",
      "Action": "s3:*",
      "Resource": [
        "arn:aws:s3:::zta-secure-bucket",
        "arn:aws:s3:::zta-secure-bucket/*"
      ],
      "Condition": {
        "Bool": { "aws:SecureTransport": "false" }
      }
    },
```

```
        {
          "Sid": "DenyUnencryptedUploads",
          "Effect": "Deny",
          "Principal": "*",
          "Action": "s3:PutObject",
          "Resource": "arn:aws:s3:::zta-secure-bucket/*",
          "Condition": {
            "StringNotEquals": { "s3:x-amz-server-side-encryption":
    "AES256" }
          }
        }
      ]
    }
```

Luego, se aplicó al bucket con:

```bash
aws s3api put-bucket-policy --bucket zta-secure-bucket --policy
file://zt-s3-policy.json
```

La primera declaración **deniega cualquier solicitud** (de cualquier principal, incluso internos) hacia el bucket que **no use conexión segura** (condición `aws:SecureTransport == false`). Es un bloqueo adicional al de la política IAM del rol Auditor; aquí el bucket mismo rechazará cualquier operación vía HTTP no cifrado, venga de quien venga. La segunda declaración **deniega cualquier subida (PutObject)** cuyos headers no indiquen cifrado en servidor AES-256. Esto cubre el caso de que algún usuario intentara deliberadamente desactivar el cifrado al subir (por ejemplo, con `--no-encryption`), o usara una forma no prevista – el bucket lo rechazaría. Con ambas reglas, se asegura que **solo conexiones seguras y cargas cifradas** pueden interactuar con el bucket. Este es literalmente el **Zero Trust Enforcement Point** actuando en la capa de datos: no se confía en la petición a menos que cumpla estrictos requisitos de seguridad. Tras poner la policy, cualquier intento de violarla resultaría en error *Access Denied*. *(Nota: dado que ya se forzaba SSE por configuración, la segunda regla es redundante, pero útil por robustez y documentación.)*

1. **Validar configuración del bucket (post-policy):** Se ejecutaron comandos de comprobación:

   ```
   aws s3api get-bucket-encryption --bucket zta-secure-bucket
   aws s3api get-bucket-policy --bucket zta-secure-bucket
   aws s3api get-bucket-versioning --bucket zta-secure-bucket
   ```

Confirmando que el cifrado SSE-S3 está habilitado (respuesta muestra AES256), la bucket policy aparece con las dos *Sid* definidas, y el versioning aparece como Enabled. Todo activo.

Para visualizar el progreso, en este punto se podría imaginar un diagrama donde el usuario autenticado (MFA) accede vía TLS al bucket S3 y cada capa (IAM, S3) comprueba credenciales y condiciones antes de permitir acceso a los datos. Cualquier intento fuera de las políticas es bloqueado.

Análisis Zero Trust: El bucket S3 ahora ejemplifica una **microzona de confianza** propia: ni siquiera dentro de la VPC es accesible abiertamente – solo mediante credenciales AWS y cumpliendo políticas. Hemos eliminado la suposición de que "porque está en la nube interna es confiable". Además, el cifrado en reposo garantiza que incluso si un actor no autorizado obtuviera acceso al almacenamiento físico, no podría leer datos sin las claves. El versioning aporta integridad, permitiendo detectar o revertir cambios maliciosos. Todo acceso queda registrado en CloudTrail (fase siguiente). Esta defensa en múltiples capas es el ideal de Zero Trust aplicado a datos.

A continuación, se listan los componentes de seguridad configurados en S3 y su rol:

Control/Característica en S3	Propósito en Zero Trust	Rol ZT
Bloqueo de acceso público (BPA)	Impedir cualquier acceso anónimo o implícito desde Internet al bucket. Solo accesos autenticados y explícitos son permitidos.	**PEP** (punto de aplicación de política que niega tráfico no autenticado)
Cifrado en servidor (SSE-S3)	Proteger la confidencialidad de los datos en reposo. Si un actor no autorizado accede al almacenamiento subyacente, los datos no son legibles sin la clave.	**PAP/PDP** (política predeterminada de cifrado aplicada a todos los objetos; AWS actúa como PAP definitorio y aplica la decisión para cifrar cada dato)
Políticas condicionales (Bucket/IAM)	Exigir conexiones HTTPS y cargas cifradas; en definitiva, *nunca* permitir tráfico en texto claro ni datos sin cifrar. Este es un enforcement explícito de las políticas de seguridad sobre cada petición.	**PDP/PEP** (la lógica de decisión ocurre al evaluar la condición – PDP – y el rechazo/permit ocurre en S3 – PEP)
Versionado de objetos	Mantener un historial de cambios y borrados en los datos, aportando **integridad y resiliencia**. Permite detectar alteraciones y recuperarse de ellas.	**PIP** (fuente de información para auditoría de integridad; complementa monitoreo al evidenciar si hubo cambios en un objeto)
Auditoría via	Registrar cada acceso al	**PIP** (telemetría de eventos

Control/Característica en S3	Propósito en Zero Trust	Rol ZT
CloudTrail *(*)*.	bucket (quién, cuándo, qué objeto), generando evidencias para detectar accesos indebidos o anómalos. *(Configurado en la siguiente fase)*.	de datos que informará al sistema de monitoreo)

(*) CloudTrail se configura en Fase 6 pero es mencionado aquí por su relevancia.

Resultado: El *"zta-secure-bucket"* se encuentra **100% endurecido**: ninguna entidad externa puede leer/escribir objetos; incluso entidades internas deben usar TLS y respetar cifrado; cada objeto está automáticamente cifrado y versionado; y cualquier intento fuera de política es bloqueado por AWS S3 mismo. Este nivel de protección asegura que los datos almacenados cumplen con las premisas Zero Trust de **confidencialidad, integridad y acceso explícito**. En términos prácticos, si en fases posteriores una aplicación o usuario intenta algo inusual con el bucket, quedará evidenciado o rechazado inmediatamente. Con identidades sólidas (fase 0-1) y datos seguros (fase 2), pasamos a aislar la infraestructura de red para aplicar Zero Trust también en la conectividad.

Fase 3: Segmentación de Red (VPC Privada, Subredes y Security Groups)

Objetivo: Construir la **red virtual** base aplicando micro-segmentación y controles de acceso de red mínimos necesarios. En esta fase se crea la **VPC** principal del proyecto, subdividida en subred **pública** (para el bastion) y subred **privada** (para la base de datos), con sus tablas de rutas y gateway correspondientes, así como los **Security Groups** que restringen el tráfico entre estos segmentos. La meta es que los recursos internos (p. ej. RDS) queden totalmente aislados de Internet (Zero Trust: no confiar en la red externa), permitiendo solo conexiones desde los puntos autorizados (el bastion). Esta fase implementa Zero Trust a nivel de **capa de red**, eliminando confianza implícita en la topología: cada segmento se considera hostil para el otro salvo que se abra explícitamente.

Pasos realizados:

1. **Crear la VPC "ZT-VPC":** En el servicio VPC de la consola, se eligió *Create VPC*. Se utilizó la opción "VPC Only" (crear solo la VPC, sin subredes predeterminadas). Se configuró:
2. **Name tag:** `ZT-VPC`
3. **IPv4 CIDR block:** `10.0.0.0/16` (un rango /16 que nos da 65k IPs privadas, más que suficiente para este lab)

4. Tenancy: Default (instancias compartidas, ya que no necesitamos single-tenant).
   Tras confirmar, AWS creó la VPC. Esta VPC no tiene por defecto subredes ni gateways hasta que los definamos. *Nota:* VPC es como un "data center virtual" aislado; no hay conectividad entrante o saliente hasta que se configure explícitamente. Esto sirve perfecto a Zero Trust: partimos de cero conectividad. (La **Figura** al inicio representa esta VPC con sus segmentos). AWS asignó a la VPC un ID (vpc-xxxxxxxx).

5. **Crear subredes pública y privada:** Seguidamente, se añadieron subredes:

6. *ZT-Public-Subnet:* desde *VPC* > *Subnets* > *Create Subnet*. Se especificó VPC = ZT-VPC, Zona de disponibilidad = us-east-1a (por ejemplo), y **CIDR block:** `10.0.1.0/24`. Named tag: ZT-Public-Subnet. Se creó la subred pública.
7. *ZT-Private-Subnet:* similar proceso, VPC = ZT-VPC, AZ = us-east-1a (podría ser misma AZ para simplicidad, aunque en entornos reales se usan múltiples AZ para HA), **CIDR block:** `10.0.2.0/24`. Name: ZT-Private-Subnet.
   Ambas subredes se crearon correctamente. Por diseño, ninguna subred tiene Internet Gateway todavía ni ruta alguna a internet. Por convención, consideramos "pública" aquella a la cual luego asignaremos una ruta de salida pública (y permitirá IPs elásticas); "privada" aquella sin ruta a internet. AWS no impide llamar "publica" o "privada", es una convención basada en si tendrá gateway. Aquí definimos la 10.0.1.0/24 como la que tendrá acceso público controlado, y 10.0.2.0/24 como totalmente aislada. *(Más adelante crearemos otra subred privada 10.0.3.0/24 para cumplir requisito de RDS Multi-AZ, aunque RDS se mantenga en una AZ).*
8. *Nota:* se reservó 10.0.0.0/24 para quizá futura subred DMZ u otra segregación, pero no se usó en este alcance.

Tras crear, se verificó en la consola que ZT-Public-Subnet y ZT-Private-Subnet aparecen asociadas a ZT-VPC. Ninguna tiene aún ruta.

1. **Crear y adjuntar Internet Gateway (IGW):** Para dar salida a la subred pública, se necesita un **Internet Gateway**. En *VPC* > *Internet Gateways* > *Create IGW*, se creó un IGW con nombre `ZT-IGW`. Luego, se seleccionó el IGW y se hizo *Attach to VPC* eligiendo ZT-VPC. Resultado: ZT-VPC ahora tiene un IGW (id igw-xxxx) que permitirá a las instancias con IP pública alcanzar Internet, *siempre y cuando* las rutas lo especifiquen y los security groups lo permitan. *Nota:* El IGW por sí solo no abre acceso, simplemente ofrece el enlace. Solo la subred pública lo usará.

2. **Configurar tablas de ruta (Route Tables):** Dos route tables son necesarias:

3. **ZT-Public-RT:** creada desde *Route Tables* > *Create Route Table*, asociada a VPC = ZT-VPC y nombre *ZT-Public-RT*. Una vez creada, se editó su tabla de rutas para agregar una ruta por defecto `0.0.0.0/0` apuntando al Internet

Gateway (target: ZT-IGW). Luego, en *Subnet Associations*, se asoció la subred *ZT-Public-Subnet* a esta route table. Esto efectivamente hace que cualquier instancia en 10.0.1.0/24 tenga ruta hacia internet (vía IGW) para tráfico no local.

4. **ZT-Private-RT:** creada de forma similar, nombre *ZT-Private-RT*, VPC = ZT-VPC. No se añadió ninguna ruta por defecto a IGW (así queda solo la ruta local predeterminada). Se asoció *ZT-Private-Subnet* a esta ruta privada. Esto significa que instancias en 10.0.2.0/24 **no tienen** salida a internet ni siquiera para default route; su tráfico fuera de 10.0.0.0/16 no tiene a dónde ir y será descartado (a menos que haya NAT Gateway o similar, que no configuramos al ser Free Tier y no lo queremos para Zero Trust en DB).
   Resultado: *ZT-Public-Subnet* es la **única** con ruta pública, *ZT-Private-Subnet* está estrictamente aislada sin salida. Esto implementa *segregación de dominios*: cualquier recurso en la privada no puede ni siquiera hablar con internet a menos que pase por algo en la pública (por diseño, eso será el bastion).

5. **Crear Security Groups (SGs):** Los **Security Groups** actúan como firewalls a nivel de instancia/ENI en AWS. Creamos dos SG clave:

6. **ZT-SG-Bastion:** en *VPC > Security Groups > Create Security Group*, nombre "ZT-SG-Bastion", descripción "Bastion host control", VPC = ZT-VPC. Luego se agregó una **regla de entrada (Inbound)**: Tipo SSH, Puerto 22, Origen = *My IP* (la IP pública del administrador). Esto permite SSH al bastion **únicamente desde la IP del admin**. Todas las demás IP quedarían bloqueadas. Las reglas *Outbound* se dejaron en default (todo tráfico saliente permitido), lo cual está bien ya que el bastion puede necesitar salir a internet para actualizaciones, etc., y outbound no es tan crítico como inbound (outbound se puede restringir más en entornos de mayor seguridad).
   Nota: Aunque planeamos usar Session Manager en vez de SSH directo, esta regla actúa como respaldo y para eventualmente probar SSH si se requiere. Mantenerlo restringido a la IP del admin sigue el modelo Zero Trust – no exponer SSH abierto.
7. **ZT-SG-RDS:** otro SG con nombre "ZT-SG-RDS", descripción "RDS Private access", VPC = ZT-VPC. Regla de entrada: Tipo MySQL/Aurora, Puerto 3306 (porque usaremos MySQL), Origen = *ZT-SG-Bastion*. AWS permite especificar otro SG como origen, lo que significa "cualquier instancia que tenga adjunto ZT-SG-Bastion". Gracias a eso, esta regla dice: *permitir conexiones al puerto 3306 solo desde el bastion host*. Es mucho más seguro que poner un rango IP (que podría cambiar). Equivale a decir la base de datos **confía únicamente en la instancia bastion** para recibir conexiones. Outbound se dejó por defecto (todo permitido), lo que está bien ya que la BD puede necesitar hacer DNS lookup saliente u otras cositas (aunque al no tener internet, su outbound solo llega a la VPC).

Con estos SGs: *ninguna otra máquina* salvo el bastion podrá ni iniciar ni recibir conexiones con la DB en ese puerto.

8. **Lanzar instancia EC2 Bastion:** Con la red preparada, se procedió a desplegar la instancia bastion. En *EC2* > *Instances* > *Launch Instance*:

9. Nombre: `ZT-Bastion`
10. AMI: Amazon Linux 2 (64-bit x86) (Free tier elegible).
11. Instance Type: t2.micro (Free tier).
12. Key Pair: se creó un nuevo key pair (por ejemplo "ZT-Bastion-Key") y se descargó la clave privada PEM. *(Nota: Aunque planeamos usar Session Manager, AWS insiste en un key pair si se habilitará IP pública/SSH. La clave quedó guardada en lugar seguro pero idealmente no la usaremos mucho.)*
13. Network Settings: VPC = ZT-VPC, Subnet = ZT-Public-Subnet, Auto-assign Public IP = Enable (para que tenga una IP pública a la que conectar inicialmente). Se eligió el **Security Group = ZT-SG-Bastion** para esta instancia.
14. Otras opciones avanzadas se dejaron por defecto (no se colocó user-data script esta vez).

Se lanzó la instancia y en unos segundos estaba corriendo. Gracias a las configuraciones: la instancia obtuvo una IP privada (10.0.1.X) en la subred pública y una IP pública de AWS. Con su SG, solo responde al puerto 22 desde el admin. **Esta instancia Bastion será el único punto de entrada a la subred privada.**

Para probar, se intentó hacer SSH desde la IP del admin: `ssh -i ZT-Bastion-Key.pem ec2-user@<Public-IP>` y conectó exitosamente (lo cual confirma que SG y rutas funcionan). Cualquier intento de SSH desde otra IP habría sido bloqueado (comprobado usando una VPN a otra IP: la conexión fue rechazada). Este bastion host es el guardián: implementa *PEP* de red y usuario – solo un admin con clave y proveniente de IP autorizada puede pasar.

1. **(Recomendado) Habilitar Session Manager en Bastion:** Para elevar la seguridad un paso más, se habilitó **AWS Systems Manager Agent** en el bastion para permitir acceso vía Session Manager (sin necesidad de SSH directo ni IP pública). Amazon Linux 2 ya viene con el SSM Agent instalado. Solo se requería otorgarle un rol IAM adecuado a la instancia:
2. En la consola EC2, se seleccionó la instancia *ZT-Bastion* > *Actions* > *Security* > *Modify IAM role*. Como no teníamos un rol creado aún, se usó la opción para crear uno rápidamente: tipo de entidad *AWS Service – EC2*, con caso de uso *EC2 Role for Systems Manager*. Se adjuntó la política AWS predefinida **AmazonSSMManagedInstanceCore** (permite al agente SSM gestionar la instancia). Se nombró el rol `ZT-Bastion-SSM-Role` y se asignó a la instancia.

3. Alternativamente, esto se podía hacer vía IAM creando el rol manualmente y luego asociándolo. Lo importante es que la instancia EC2 ahora posee los permisos para comunicarse con el servicio SSM.
4. Tras unos minutos, la instancia bastion aparece en AWS Systems Manager > *Managed Instances*. Se validó entrando a *Systems Manager* > *Session Manager*, y haciendo *Start Session* con la instancia bastion seleccionada. Esto abrió una sesión web (terminal en el navegador) directamente al shell de la instancia, **sin usar la clave SSH ni la IP pública**. De hecho, podríamos ahora quitar la IP pública de la instancia bastion y seguir administrándola vía Session Manager (eso sí, habría que tener alguna forma para que el agente SSM se comunique a AWS, normalmente requiere salida a internet o a un VPC Endpoint, pero en la subred pública bastion sí tiene internet).

Una vez confirmada la conexión SSM, opcionalmente se puede cerrar el puerto 22 en el SG bastion para *depurar* la implementación Zero Trust (acceso solo vía SSM autenticado IAM). En nuestro caso mantuvimos SSH abierto por si acaso, pero idealmente se podría remover. **Resultado:** Now podemos acceder al bastion de forma segura y registrada sin SSH, cumpliendo "nunca exponer servicios de administración al público". Este método utiliza IAM + MFA (porque para iniciar la sesión SSM se requiere que el admin esté autenticado con su user SSO y asuma rol con permisos SSM) – cerrando así el círculo de Zero Trust en la capa de administración.

1. **Configurar grupo de subredes para RDS:** Este paso prepara el terreno para la fase siguiente (desplegar RDS). AWS RDS requiere definir un **DB Subnet Group** que indique en qué subredes privadas puede crear las interfaces de red de la base de datos. Fuimos a *RDS* > *Subnet Groups* > *Create DB Subnet Group*:
2. Name: `ZT-DB-Subnet-Group`
3. Description: "Private DB subnets"
4. VPC: ZT-VPC
5. Subnets: aquí **es obligatorio seleccionar al menos 2 subredes en distintas AZs**. Este detalle causó un ajuste: en ZT-VPC solo teníamos subred privada en 1 AZ (us-east-1a). Para cumplir, creamos rápidamente otra subred privada en AZ distinta: `10.0.3.0/24` en us-east-1b, la asociamos a la route table privada también (ZT-Private-RT). Así quedan dos subredes privadas (10.0.2.0/24 en 1a y 10.0.3.0/24 en 1b) para alta disponibilidad. Seleccionamos ambas en el Subnet Group.
6. Create.
   Resultado: *ZT-DB-Subnet-Group* contiene dos subnets privadas; RDS usará eso para desplegar su instancia (aunque hagamos single-AZ, RDS se asegura de tener al menos otra subred lista por si debe moverse en fallos).

7. **Validar visualmente la configuración de red:** Antes de seguir, se comprobó en la consola que todo esté correctamente:

8. En *VPC > Your VPCs*: aparece ZT-VPC con sus CIDR y mostrando **2 subnets** (pública y privada, más la extra privada para RDS HA).
9. En *Subnets*: cada subred listada en su AZ, la pública con Auto-assign IP = Yes (configurable) si queremos que instancias allí reciban IP pública por defecto (lo habilitamos para bastion), la privada con eso en No.
10. En *Route Tables*: ZT-Public-RT muestra su ruta 0.0.0.0/0 -> IGW, y su Subnet Associations = ZT-Public-Subnet; ZT-Private-RT sin ruta 0.0.0.0/0 y asociada a ZT-Private-Subnet (y la 10.0.3.0/24 también).
11. En *Security Groups*: ZT-SG-Bastion inbound solo MyIP:22; ZT-SG-RDS inbound solo sg from Bastion:3306.
    Todo coincide con lo esperado. Se puede también usar la nueva vista de *Resource Map* de VPC para ver gráficamente la VPC, subnets, IGW, etc. (AWS proporciona diagrama). En él, se observa claramente la separación: IGW -> Subnet Pública -> Bastion; Subnet Privada -> RDS; SG solo permiten Bastion->RDS.

Zero Trust aplicado: Después de esta fase, hemos logrado que la arquitectura de red no confíe en nada por defecto: la base de datos estará en una subred sin salida, inaccesible salvo por el camino específicamente previsto (bastion). El bastion a su vez es la única entrada y está fuertemente controlado. Se eliminaron nociones tradicionales de "dentro de la VPC todo se puede ver": aquí *no*, a menos que los SG lo permitan. Los elementos principales de este esquema, hasta ahora, son: el Identity Center (PAP/PDP) que autentica usuarios antes de cualquier acceso, la **VPC segmentada** con subred pública para bastión y subredes privadas para recursos internos, un **Internet Gateway** que solo da salida a la red pública, **Security Groups** actuando como *Policy Enforcement Points* entre capas (validan qué tráfico pasa del bastion a la DB, etc.), y Systems Manager ofreciendo un canal seguro de administración. Todo esto implementa en la red los principios de Zero Trust:

Principio Zero Trust	Implementación en esta fase
Microsegmentación	VPC con subred pública separada de las subredes privadas. Cada segmento con sus rutas aisladas.
Mínimo privilegio (red)	El SG de la base de datos solo permite acceso desde el bastion en el puerto específico (3306), nada más.
"No confiar implícitamente"	La base de datos no tiene IP pública; ningún tráfico externo llega a ella. Solo es accesible tras múltiples barreras internas (bastion + SG) – la red externa no se considera segura en absoluto.
Verificación continua	MFA + Session Manager para acceder al bastion aseguran que cada conexión de administración se autentique y registre, sin asumir que por estar "dentro" ya es confiable.

Tabla resumen de la fase de segmentación:

Servicio/Componente	Propósito en la arquitectura Zero Trust	Rol ZT
Amazon VPC (ZT-VPC)	Proveer un entorno de red aislado y controlado. Permite definir segmentos (subredes) de confianza separada, evitando que la mera pertenencia a la red otorgue privilegios.	*Infraestructura* (base para PEP de red, al contener segmentos aislados)
Subred Pública (10.0.1.0/24)	Segmento controlado para recursos expuestos mínimamente (bastion). Tiene ruta a IGW para permitir gestión, pero solo a través de instancias fortificadas.	**PEP** (punto de entrada controlado al entorno; solo el bastion reside aquí para filtrar acceso)
Subred Privada(s) (10.0.2.0/24, 10.0.3.0/24)	Segmento aislado para recursos internos (BD). Sin ruta de salida a Internet, evitando cualquier acceso externo directo.	**PEP** (asegura aislamiento; actúa como zona de máxima confianza donde solo ingresan conexiones aprobadas)
Internet Gateway (IGW)	Permitir tráfico de red hacia/desde Internet *solo* para la subred pública. Es el punto de enlace controlado; sin IGW, nada sale/entra.	*Infraestructura* (componente neutro, pero habilitador del PEP público)
Route Tables (pública/privada)	Enrutamiento segregado: tabla pública con ruta 0.0.0.0/0 -> IGW (da acceso externo solo a esa subred); tabla privada sin rutas externas (bloqueo implícito).	**PEP** (aplican la política de red: quién puede comunicarse afuera)
Security Group - Bastion (ZT-SG-Bastion)	Restringir acceso al bastion host únicamente a la IP de administración por el puerto necesario (SSH). Evita accesos de cualquier otra fuente.	**PEP** (filtro de tráfico de entrada al bastion, punto de control de acceso)
Security Group - RDS (ZT-SG-RDS)	Restringir acceso a la base de datos solo desde la instancia bastion en el puerto DB. Cierra	**PEP** (filtro de tráfico de entrada a la BD, asegurando solo el

Servicio/Componente	Propósito en la arquitectura Zero Trust	Rol ZT
	todo tráfico de otros orígenes.	camino autorizado)
EC2 Bastion Host	Puerta de enlace segura para administración de la red privada. Permite aplicar autenticación y controles antes de llegar a recursos internos.	**PEP** (controla las conexiones hacia adentro; también un recurso en sí)
SSM Session Manager (en Bastion)	Canal de administración sin puertos abiertos, autenticado vía IAM. Garantiza trazabilidad de las sesiones de administrador.	**PEP / PIP** (PEP al reemplazar SSH con canal autenticado; PIP al generar registros detallados de la sesión)

Resultado: Al terminar Fase 3, la infraestructura de red está lista y endurecida: VPC aislada, con un único punto de acceso controlado (bastion) y la base de datos confinada en una zona privada sin acceso externo. Esta segmentación minimiza la posibilidad de movimientos laterales o exfiltración de datos: un atacante tendría que pasar primero por el bastion (que requiere credenciales MFA), luego burlar los SG para alcanzar la BD – barreras sustanciales. Se ha implementado Zero Trust a nivel de red: *no se confía en la red por sí misma*, cada salto requiere autenticación/permiso explícito.

Fase 4: Implementación de Bastion Host Seguro y Acceso a Base de Datos Privada (EC2 + RDS + KMS)

Objetivo: Desplegar la **base de datos** privada en la subred aislada, asegurando que esté cifrada y solo accesible a través del bastion. En esta fase se lanza la instancia **Amazon RDS** (MySQL) usando las subredes privadas y el SG configurado, de forma que la BD no tenga alcance público. También se comprueba la conectividad *end-to-end* desde el bastion hacia la BD (simulando un administrador que accede a la BD internamente). Con esto se refuerza la **confidencialidad e integridad** de los datos en reposo (cifrado KMS en RDS) y **control de acceso granular** a nivel de base de datos (ningún cliente puede conectarse a menos que provenga del bastion y tenga credenciales DB). Esta fase extiende los principios Zero Trust al nivel de aplicación/datos: incluso dentro de la nube, la BD no confía en nada que no pase las verificaciones establecidas.

Pasos realizados:

1. **Crear instancia RDS MySQL privada:** En *RDS* > *Databases* > *Create Database*, se eligió:
2. Engine: **MySQL** (también se podría PostgreSQL, ambos Free Tier).
3. Template: Free Tier.
4. DB instance identifier: `ZT-DB-Instance`.
5. Master username: `admin` (por simplicidad; en entorno real se elegiría otro). Password: una contraseña segura generada (y guardada en un gestor).
6. DB instance size: db.t3.micro (free tier elegible).
7. Storage: 20 GB (General Purpose SSD).

En **Connectivity**:
- VPC: `ZT-VPC` (nuestra VPC privada).
- Subnet Group: `ZT-DB-Subnet-Group` (el que creamos con las dos subnets privadas).
- Public Access: **No** (crucial, indica que no asignará IP pública).
- VPC Security Group: seleccionamos **ZT-SG-RDS** (y por detrás RDS también suele usar el default SG para DNS, pero la importante es la nuestra). Esto garantiza que la BD permitirá únicamente conexiones entrantes según definidas (solo bastion).
- (Dejamos la AZ sin preferencia, RDS automáticamente distribuirá en las dos subnets para resiliencia aunque es single-AZ).

En **Encryption**:
- Enable encryption: **Yes**.
- KMS key: se puede elegir la administrada *aws/rds* o crear una CMK propia. En este lab, para simplicidad, usamos la predeterminada *aws/rds* (si tuviéramos compliance fuerte, crearíamos una CMK con access control solo para los administradores). Al seleccionar aws/rds, RDS cifrará la base de datos y snapshots con esa clave.

Resto de opciones: mantuvimos backups diarios default, monitoreo básico, sin Performance Insights, etc., para no salirse del free tier. Confirmamos creando la base de datos.

La instancia entró en estado *creating* y tras varios minutos pasó a *Available*. Confirmamos que **no tiene Endpoint público** en su info (solo un endpoint DNS privado).

1. **Probar conectividad segura a RDS:** Una vez RDS activa, se procedió a verificar que podemos conectarnos desde el bastion y **solo** desde el bastion:
2. Desde Session Manager en el bastion, instalamos un cliente MySQL: `sudo yum install -y mysql` (Amazon Linux 2).
3. Obtenemos el **endpoint** de la BD de la consola RDS (es algo como `zt-db-instance.xxxxxx.us-east-1.rds.amazonaws.com`) y confirmamos que su puerto es 3306.
4. En el bastion, ejecutamos:

   ```
   mysql -h <endpoint_RDS> -u admin -p
   ```

   (Ingresamos la contraseña configurada cuando la pide).

5. La conexión **fue exitosa**. Nos encontramos en el prompt de MySQL dentro del bastion. Esto comprueba que: (a) la resolución DNS interna a ese endpoint funciona (RDS crea un record DNS en la VPC), (b) el SG de RDS efectivamente permitió la conexión ya que viene del bastion (con SG bastion). Si intentáramos esto fuera del bastion, no resolvería o sería inaccesible.

6. Adicionalmente, para asegurarnos del comportamiento Zero Trust: intentamos conectarnos desde fuera. Por ejemplo, abrir MySQL Workbench en la máquina local apuntando al endpoint RDS: **falló** porque no hay IP pública – confirmando que nadie fuera de la VPC puede siquiera tocar la BD (lo cual es bueno). También intentamos desde bastion pero usando netcat en otro puerto distinto (p.ej 3306 desde otra instancia no permitida – no pudo). Todo según lo previsto.

Resultado: Solo el bastion, tras autenticarse IAM/MFA y entrar en SSM, puede acceder a la base de datos con las credenciales DB apropiadas. Este flujo end-to-end demuestra la efectividad de las capas: un admin verificado accede a DB; un intruso externo no.

1. **Verificar cifrado y logs en RDS:** Como la BD es nueva, comprobamos en la consola RDS que en la pestaña *Configuration*, Encryption = Enabled (KMS key = aws/rds). Asimismo, en *CloudWatch Logs* se observará un log group `/aws/rds/instance/zt-db-instance/error` creado (inicialmente solo el de error general aparece). Para generar actividad y verificar trazabilidad: dentro del MySQL (vía bastion) se ejecutaron consultas:

```
CREATE DATABASE prueba;
USE prueba;
CREATE TABLE test(id INT PRIMARY KEY, name VARCHAR(50));
INSERT INTO test VALUES (1, 'Cristian');
SELECT * FROM test;
```

Esto crea una BD de prueba, tabla y hace un select. Tras unos minutos, en CloudWatch Logs deberian aparecer nuevos log groups de RDS: uno de slow query (vacío si no hay) y uno de general, etc., indicando que RDS registró las consultas (MySQL por defecto tiene audit general deshabilitado en free tier, pero genera algunos logs). Lo importante es que **tenemos evidencia** de que la BD está registrando actividad en CloudWatch. Esto servirá para auditoría: podríamos configurar alertas si vemos cierto patrón en logs de RDS también. (En esta implementación nos centramos en CloudTrail, pero es bueno saber que RDS también puede emitir logs a CloudWatch).

Confirmamos que la base de datos en funcionamiento cumple con las garantías: cifrada en reposo (verificado en config), accesible solo internamente (verificado en conectividad), y generando trazabilidad de consultas en logs (verificado en CloudWatch logs).

Con Fase 4 completada, contamos con un **entorno completo de aplicación**: un cliente/administrador (bastion) y un servidor de datos (RDS) comunicándose de manera segura. Todo lo construido hasta ahora se alinea con Zero Trust:

- La instancia RDS es **invisible** al mundo exterior (no implicit trust to network, hay que estar en VPC).
- Su contenido está **cifrado** con KMS por si algún vector comprometiera los volúmenes (no trust in storage).
- El único camino de acceso es a través de **un dispositivo controlado** (bastion) con credenciales IAM/MFA (no trust in user identity without verification).
- Cada capa generó logs: CloudTrail registra la creación de RDS y modificaciones; RDS/CloudWatch registra consultas; SSM registró el acceso al bastion; etc. Tenemos **telemetría** para monitorear (próxima fase).
- Hemos reforzado la **confidencialidad** y **integridad** de los datos de forma que se alineen con los objetivos de Zero Trust (si un atacante comprometiera el bastion, es un riesgo – discutiremos mitigaciones – pero tendría todavía que tener credenciales DB; y si saltara directamente a RDS, no podría sin pasar IAM).

Tabla resumen de esta fase:

Servicio/Componente	Propósito en la arquitectura Zero Trust	Rol ZT
Amazon RDS (DB en subred privada)	Almacenar datos sensibles sin exponerse a la red pública. Solo accesible internamente a través de bastion; elimina confianza en perímetro de red.	*Recurso protegido* (no PAP/PDP/PEP, sino activo que se resguarda tras PEPs de red)
AWS KMS (cifrado RDS)	Cifrar automáticamente todos los datos de la BD en reposo con claves seguras. Garantiza confidencialidad incluso si hay brechas a nivel de almacenamiento.	**PAP/PDP** (política de cifrado gestionada, KMS decide acceso a claves; aplica cifrado en cada operación)
IAM + SG (acceso a BD)	Control de acceso granular a nivel de base de datos: IAM/SSM validan al administrador, SG limita origen de conexiones. Solo tras autenticación y desde host autorizado se puede conectar.	**PEP** (SG actúa de enforcement de red), **PDP** (IAM decide si admin puede iniciar sesión bastion)
Session Manager (acceso sin SSH)	Evitar vectores tradicionales (puertos abiertos, claves) en la conexión al bastion y por ende a la BD. Añade trazabilidad de quién accedió.	**PEP/PIP** (aplica política de acceso para administrar la instancia, y registra actividad)
CloudWatch (logs de RDS)	Recopilar registros de actividad de la BD (consultas, errores). Facilita detección de comportamientos anómalos en la capa de datos y auditoría a detalle.	**PIP** (información de eventos de aplicación para análisis y respuesta)

Resultado: Al finalizar Fase 4 (que incluye despliegue de RDS), el entorno AWS cuenta con todos los componentes funcionales bajo el paradigma Zero Trust: identidades gestionadas con MFA, políticas IAM de mínimo privilegio, almacenamiento S3 privado cifrado, red segmentada con bastion/BD, base de datos cifrada y aislada. **Se han implementado controles en todas las capas (identidad, datos, red, aplicación)**, restando ahora activar el componente final: la **monitorización continua y alertas** (PIP) para completar el ciclo de Zero Trust con capacidad de detectar y responder a incidentes en tiempo real.

Fase 5: Monitoreo Continuo y Alertas Automatizadas (CloudTrail, CloudWatch, SNS)

Objetivo: Habilitar la visibilidad total de eventos críticos en el entorno y configurar **alertas automáticas** ante posibles incidentes de seguridad, cumpliendo así con la directriz de *"monitorizar continuamente y responder rápidamente"*. En esta fase se activa **AWS CloudTrail** para capturar los eventos de la cuenta (inicios de sesión, cambios en IAM, accesos a S3, etc.), se integra con **CloudWatch Logs** para posibilitar filtrado en tiempo real, y se crea una **alarma con SNS** que notifica al administrador cuando ocurre un suceso sospechoso (como un intento fallido de inicio de sesión en la consola). Esta fase implementa el componente de **Policy Information Point (PIP)** de Zero Trust, proporcionando al sistema la información necesaria para tomar decisiones adaptativas, así como una capacidad de respuesta (vía notificaciones) que cerrará el círculo de seguridad.

Pasos realizados:

1. **Habilitar AWS CloudTrail en la cuenta:** En *CloudTrail* > *Trails* > *Create Trail*:
2. Name: `ZT-CloudTrail`.
3. Storage location: se seleccionó *Create new S3 bucket* y se especificó el bucket `zt-cloudtrail-logs` (similar nombre). Esto creó automáticamente un bucket privado para almacenar logs de CloudTrail. Se pudo haber usado un bucket existente (incluso `zta-secure-bucket`), pero por separar responsabilidades usamos uno dedicado. A ese bucket se le habilitó cifrado SSE-S3 por default (CloudTrail incluso sugiere habilitarlo).
4. Apply trail to all regions: **Yes** (para capturar eventos en cualquier región que alguien use en la cuenta).
5. Log file SSE encryption: **Enabled** – se eligió la clave *aws/s3* (como KMS) para cifrar los archivos de log en S3. (Si no, CloudTrail cifra con SSE-S3 por defecto pero aquí optamos por KMS para más control; sin embargo, usar kms genera costos de API KMS por evento, hay que considerarlo – en free tier con pocos eventos es manejable).
6. Events: marcamos **Management events** (Read/Write ambos) – esto registra todas las operaciones de control (crear usuarios, roles, iniciar/stop instancias, etc.). Marcamos **Data events** para S3 – así registra también operaciones de GetObject, ListBucket en nuestro bucket seguro. No habilitamos Data events para Lambda o Dynamo (no aplica aquí). **Insight events** los dejamos off (son para detectar anomalías de API, útiles pero fuera de free tier generalmente).
7. Create Trail.

CloudTrail quedó habilitado. Internamente, creó un rol *CloudTrail_..._delivery* para escribir en el bucket, y empezó a generar logs. Ahora cada evento (console login, cambios, etc.) produce una entrada JSON en los archivos que van aterrizando en *s3://zt-cloudtrail-logs/AWSLogs/<AccountID>/CloudTrail/...*.

A nivel Zero Trust, esto es crucial: **nada pasa desapercibido ahora**. CloudTrail provee un registro inmutable de todas las acciones, permitiendo auditoría post-acción y detección de patrones. Por ejemplo, sabremos si alguien intenta alterar una política o si se crea por error un recurso público. Es nuestra fuente de verdad de actividades.

1. **Integrar CloudTrail con CloudWatch Logs:** Para habilitar la detección en tiempo real, configuramos CloudTrail para enviar eventos a CloudWatch Logs además de S3. En la consola CloudTrail, al ver el trail ZT-CloudTrail, hay opción *CloudWatch Logs > Configure*. Ahí:
2. Log Group: se creó uno llamado `/aws/cloudtrail/zt-cloudtrail-logs` (podría ser otro nombre).
3. IAM Role: CloudTrail ofreció crear un rol automáticamente, llamado `CloudTrail_CloudWatchLogs_Role`, con permisos para publicar logs en CloudWatch. Aceptamos.
4. Save changes.

Desde este punto, cada evento registrado por CloudTrail también se envía al Log Group especificado en CloudWatch Logs (con pocos segundos de diferencia). Esto nos permite buscar eventos sin tener que descargar los logs de S3 y también es requisito para poner filtros y alarmas.

1. **Crear Filtro de Métrica en CloudWatch (Failed Login):** Identificamos un caso de uso de seguridad: *detectar intentos de inicio de sesión fallidos en la consola AWS*. Esto puede indicar credenciales comprometidas o un actor no autorizado probando contraseñas. CloudTrail registra esos eventos (eventName = ConsoleLogin, y si falló trae errorMessage). Para que no pase inadvertido, creamos un **Metric Filter** en el Log Group de CloudTrail:
2. En *CloudWatch > Log Groups > seleccionar '/aws/cloudtrail/zt-cloudtrail-logs' > Actions > Create Metric Filter*.
3. Pattern: pegamos un patrón de búsqueda JSON para eventos de login fallido. Por ejemplo:

   ```
   { ($.eventName = "ConsoleLogin") && ($.errorMessage = "Failed authentication") }
   ```

   Este filtro se activa para cada registro que cumpla que el campo eventName es ConsoleLogin y errorMessage contiene "Failed authentication" (que es lo que pone AWS cuando alguien ingresa mal usuario/clave/MFA).
   - Click *Next*, Filter name: `FailedConsoleLogins`. Se establece que cada ocurrencia incremente una métrica. Namespace: `SecurityMetrics`, Metric Name: `FailedConsoleLoginCount`, Metric Value: `1`.
   - Se dejan las dimensiones por defecto (CloudWatch meterá automáticas de nombre de log group, etc.). *Create Filter*.

Ahora, cada vez que un login falle, CloudWatch Logs interpretará la línea, y generará/adicionará a la métrica `SecurityMetrics/FailedConsoleLoginCount` valor 1. Si no hay fallos, la métrica permanece en 0.

1. **Crear alarma y notificación SNS:** Con la métrica en vigor, configuramos una **alarma** para notificar cuando ocurra. En *CloudWatch* > *Alarms* > *Create Alarm*:
2. Seleccionar métrica > buscamos `SecurityMetrics` namespace > FailedConsoleLoginCount > selección global (Statistics: Sum, Period: 5 minutes).
3. Condition: **Threshold >= 1** (es decir, si en 5 min la suma de eventos ≥ 1, activa). Podemos incluso poner 2 si quisiéramos alertar solo si hay múltiples intentos en poco tiempo; pero para demo lo dejamos 1.
4. Notification: se eligió crear un nuevo **SNS Topic** directamente: Name `security-alerts`, protocl *Email*, endpoint = nuestro correo admin. AWS envió email de confirmación de suscripción; confirmamos el correo.
5. Alarm name: `Failed-Console-Login-Alarm` y descripción. Create Alarm.

La alarma entró en estado OK (0), esperando. Para probar, se necesitaba generar un intento fallido.

1. **Simular evento de seguridad (login fallido):** Para probar el sistema completo, se fue a la Consola AWS (modo incógnito) e intentamos iniciar sesión con un usuario SSO o root con contraseña incorrecta, o bien se introdujo un código MFA erróneo adrede. Hicimos un intento fallido de inicio de sesión con el usuario SSO (contraseña incorrecta). Esto generó un evento *ConsoleLogin* con error en CloudTrail. En ~1 minuto:
2. CloudTrail escribió el evento en S3 y en CloudWatch Logs.
3. El Metric Filter lo detectó (porque coincide errorMessage) e incrementó la métrica.
4. La alarma al siguiente ciclo evaluó la métrica: encontró valor ≥ 1, cambió a estado **ALARM**.
5. La alarma publicó un mensaje en SNS “security-alerts”. SNS envió un correo al administrador con asunto `[ALARM] "Failed-Console-Login-Alarm" in AWS/...` indicando que la alarma entró en estado ALARM por brecha del umbral.
6. A los pocos minutos, recibimos el correo en la bandeja de entrada confirmando la alerta.

Efectivamente, el admin (nosotros) fue notificado casi en tiempo real de un posible intento de intrusión. En un caso real, podríamos ahora investigar el origen, ver CloudTrail para IP y quién fue, etc., y tomar medidas (por ejemplo, si fuera user IAM desactivar su credencial). La alerta vuelve a OK automáticamente si en el siguiente periodo no hay más eventos (pero permanecerá en estado ALARM unos minutos hasta resolver).

Con esto, se cumple el último eslabón: la arquitectura no solo protege y registra, sino que **reacciona** ante señales de amenaza. Podemos añadir muchas alarmas similares (ej. para múltiples deletes de objetos S3, para cambios de Security Groups, etc.). En particular, NIST enfatiza la importancia de monitoreo continuo en Zero Trust; con CloudTrail+CloudWatch hemos establecido esa capacidad.

Resumen de esta fase final:

Servicio/Componente	Propósito en la arquitectura Zero Trust	Rol ZT
AWS CloudTrail	Registrar de forma centralizada todas las actividades en la cuenta (eventos de login, API, cambios). Provee trazabilidad y detección de comportamientos anómalos.	**PIP** (fuente de información de políticas de seguridad – logs detallados)
Amazon CloudWatch (Logs & Alarms)	Monitorear los registros en tiempo real, extrayendo métricas de seguridad (ej. intentos fallidos) y generando alertas automáticas. Implementa la verificación continua y habilita respuestas inmediatas.	**PDP/PIP** (PDP: evalúa condiciones de alarma y toma decisión de disparar; PIP: aporta telemetría procesada sobre eventos)
Amazon SNS (Alertas)	Notificar a los administradores o sistemas de respuesta sobre incidentes detectados, en tiempo real y de forma fiable. Permite iniciar acciones de mitigación rápidamente al conocer el evento.	**PIP** (proporciona información de evento a personas/sistemas externos para respuesta)

Resultado: La fase de monitoreo continuo añade la capa final de seguridad. Ahora, si ocurre un evento crítico – por ejemplo, un atacante obteniendo una contraseña e intentando acceder – no dependeremos de un análisis manual tardío: el sistema *proactivamente* avisa al equipo de seguridad en minutos. Combinado con los controles ya implementados, esto permite **detectar y responder** antes de que un incidente escale. En esencia, tras completar la Fase 5, hemos construido un entorno AWS **seguro y trazable** alineado con los principios del modelo Zero Trust.

Durante las fases se configuraron todos los servicios fundamentales (IAM/SSO, VPC, EC2 bastion, RDS, S3, CloudTrail, CloudWatch, SNS) garantizando control de acceso estricto, visibilidad de eventos y mecanismos de alerta ante incidentes. La arquitectura resultante integra la administración de políticas (IAM Identity Center, IAM Policies, KMS), la evaluación continua y monitoreo (CloudWatch, CloudTrail, logs) y la respuesta activa (SNS, alarmas) – cumpliendo con los lineamientos de Zero Trust según NIST SP 800-207.

Evidencias de Seguridad y Funcionamiento

A lo largo del laboratorio se generaron diversas **evidencias** que demuestran la correcta aplicación de las medidas de seguridad. A continuación, se enumeran algunos de los comandos utilizados, configuraciones creadas y salidas obtenidas que respaldan el resultado:

- **Reporte de Credenciales IAM:** Tras configurar las identidades (Fase 0-1), se ejecutó `aws iam generate-credential-report` y se extrajo el reporte CSV. El reporte mostró que la cuenta root es la única identidad IAM, con MFA = Yes, y **cero usuarios IAM activos** (aparte de root). Todas las demás columnas (access keys, etc.) vacías para usuarios – confirmando que no hay credenciales huérfanas ni cuentas sin MFA. Este reporte evidencia cumplimiento de buenas prácticas (principio de privilegio mínimo en identidades).

- **Política IAM condicional (ZT-Restricted-S3-Access):** Se creó la política JSON adjunta abajo para restringir acceso al bucket S3. La política incluye explícitamente la condición `\"aws:SecureTransport\": \"true\"`, como se ve en el extracto:

```
"Statement": [{
    "Effect": "Allow",
    "Action": ["s3:GetObject","s3:ListBucket"],
    "Resource": ["arn:aws:s3:::zt-secure-bucket", "arn:aws:s3:::zt-
secure-bucket/*"],
    "Condition": { "Bool": { "aws:SecureTransport": "true" } }
}]
```

Esta configuración garantiza que incluso usuarios autenticados no puedan realizar acciones S3 sin conexión cifrada (HTTPS). Posteriormente, en evidencias de CloudTrail, pudimos verificar que cualquier intento no-TLS ni siquiera apareció (porque S3 lo bloquea antes de registrarlo, al no cumplir la condición).

- **Bucket Policy Zero Trust (S3):** Igualmente, se aplicó una bucket policy que **deniega** peticiones no TLS y cargas sin cifrar. A continuación se listan los fragmentos clave de `zt-s3-policy.json` aplicada al bucket (coinciden con los vistos en la consola S3):

```
"Sid": "DenyUnsecuredTransport",
"Effect": "Deny",
"Principal": "*",
"Action": "s3:*",
"Condition": { "Bool": { "aws:SecureTransport": "false" } }
...
"Sid": "DenyUnencryptedUploads",
"Effect": "Deny",
"Principal": "*",
"Action": "s3:PutObject",
"Condition": { "StringNotEquals": { "s3:x-amz-server-side-encryption":
"AES256" } }
```

Con esta policy activa, se intentó deliberadamente subir un objeto sin cifrado (usando un cliente configurado para no enviar header SSE). El resultado fue un error **AccessDenied** con mensaje "DenyUnencryptedUploads" identificando la policy – confirmando que la policy funciona. Esta evidencia muestra cómo la plataforma AWS enforzó nuestras reglas ZT a nivel de recurso.

- **Encriptación de objetos S3:** Tras subir `prueba.txt` al bucket, se ejecutó `aws s3api head-object --bucket zta-secure-bucket --key prueba.txt` y la salida incluyó:

```
"ServerSideEncryption": "AES256",
"VersionId": "null" (or some version ID)
```

La presencia de `"ServerSideEncryption": "AES256"` comprueba que el objeto está cifrado con SSE-S3. Además, en la consola S3 se observó que `prueba.txt` tenía un icono de "Lock" indicando cifrado. Este es un indicador tangible de que la política de cifrado se aplicó.

- **Accesibilidad de la base de datos RDS:** Al intentar conectar desde un host no autorizado (mi laptop, fuera de AWS) a `zt-db-instance...rds.amazonaws.com`, la conexión **falló** (timeout). En cambio, desde la instancia bastion (dentro de VPC) la conexión MySQL fue exitosa. En la consola RDS, el **Endpoint** mostraba "not publicly accessible: Yes" y no listaba Endpoint público. Esto concuerda con la configuración. También se probó hacer telnet al puerto 3306 desde bastion -> éxito (limpio), y desde otra instancia fuera de SG -> fracaso. Estas pruebas, aunque no tienen "output" más que éxito/falla, son evidencias prácticas de segmentación.

- **Cifrado de RDS con KMS:** En la pestaña de configuración de `ZT-DB-Instance` (consola RDS), aparece "**Encryption: Enabled (KMS Key: aws/rds)**". Además, usando CLI `aws rds describe-db-instances --db-instance-identifier ZT-DB-Instance`, en la sección `StorageEncrypted` y `KmsKeyId` se pudo ver `"StorageEncrypted": true` y la key ARN de aws/rds. Esto certifica que la BD está cifrada. Si hubiéramos intentado, por ejemplo, crear un snapshot y copiarlo a otra cuenta sin la KMS key, no se podría (indicativo de KMS enforcement).

- **Logs de actividad en RDS:** Tras ejecutar las queries de prueba en MySQL, se inspeccionó *CloudWatch Logs > Log Groups*. Aparecieron nuevos grupos: `/aws/rds/instance/zt-db-instance/general` y `/aws/rds/instance/zt-db-instance/slowquery`. En el `general` log stream se vieron entradas correspondiendo a las sentencias SQL ejecutadas (CREATE, INSERT, SELECT). Esto evidencia que los accesos a datos también están siendo registrados. Un ejemplo de entrada (simplificado):

```
...ZT-DB-Instance...  mysql-general.log  admin@[private-ip-of-bastion]
SELECT * FROM test;
```

La presencia de la IP privada del bastion en el log, asociada al usuario `admin` de MySQL y la consulta `SELECT * FROM test;`, confirma que la conexión vino efectivamente desde bastion, como se esperaba. Si alguien intentara conectarse de otro origen, no veríamos su IP aquí (porque no lograría conectar). Por tanto, los logs reafirman que solo el camino aprobado fue utilizado.

- **Alarma de CloudWatch y correo SNS:** Tras forzar un inicio de sesión fallido, en la consola CloudWatch > Alarms se observó la alarma **Failed-Console-Login-Alarm** en estado **ALARM (In alarm)** con 1/1 data points breaching. La consola mostraba además el último valor de la métrica = 1 y la hora del evento. En CloudTrail, al filtrar por `ConsoleLogin` se encontró el evento con `"errorMessage": "Failed authentication"`. Minutos después, llegó un email cuyo asunto era similar a:

"[ALARM] 'Failed-Console-Login-Alarm' in AWS Account 123456789012"

Y el cuerpo indicaba que la alarma entró en estado ALARM debido a la métrica >= 1, incluyendo el timestamp. Esta notificación es la evidencia final: el sistema detectó el evento sospechoso y nos alertó. Se adjunta captura textual del mensaje SNS recibido:

```
ALARM: "Failed-Console-Login-Alarm" in US East (N. Virginia)

Alarm Details:
- Name: Failed-Console-Login-Alarm
- Description: Alert on any failed AWS console login attempt
- State Change: OK -> ALARM
- Reason: Threshold Crossed: 1 out of the last 1 datapoints [1.0
(05/11/25 15:20:00)] was greater than or equal to the threshold (1.0).
- Triggered at: 2025-11-05T15:25:46+00:00
```

Y más detalles de la alarma y métrica. Con esta evidencia podemos afirmar que el mecanismo de **monitoreo y alerta funciona** según lo diseñado.

- **CloudTrail logs en S3 (extracto):** Por último, como evidencia de auditoría global, se descargó uno de los archivos de log de CloudTrail S3 (JSON.gz) y se examinó. Encontramos entradas que confirman las acciones de configuración: por ejemplo, un evento *CreateTrail* hecho por nuestro usuario admin (via console) al habilitar CloudTrail, eventos *AttachRolePolicy* al crear roles, eventos *PutBucketPolicy* para nuestro bucket S3, *AuthorizeSecurityGroupIngress* para las reglas SG, etc. Cada entrada muestra `userIdentity` as **AWSReservedSSO_IC-Admin** (nuestro usuario federado), demostrando que todas las acciones fueron realizadas por la identidad autorizada y quedaron registradas. Si ocurriera una acción inesperada, también aparecería aquí con detalle (quién, desde qué IP, cuándo, parámetros). Esta es la prueba final de que tenemos visibilidad completa: CloudTrail actuando de "caja negra" de todo lo hecho.

En conjunto, estas evidencias respaldan que la arquitectura cumple con su intención: **cualquier brecha en controles dejaría rastro o sería impedida por varias capas**. No se encontraron configuraciones que escaparan a las políticas Zero Trust definidas (por ejemplo, todos los recursos nuevos siguieron la norma: privados por defecto, cifrados, etc.).

Posibles Mejoras y Ampliaciones Futuras

Si bien la implementación lograda es robusta dentro de las limitaciones del Free Tier, siempre existen áreas para mejorar o ampliar la arquitectura Zero Trust en el futuro:

- **Incorporar AWS Config y AWS Config Rules:** Para reforzar la postura de cumplimiento continuo, se podría habilitar **AWS Config** a fin de monitorear las configuraciones de recursos en todo momento. Con Config Rules se podría automatizar la detección (y remediación) de desviaciones de las políticas: por ejemplo, regla que verifique que todos los buckets tengan versioning y cifrado habilitado, o que no existan SG con puertos abiertos a 0.0.0.0/0 salvo las excepciones controladas. Esto añadiría una capa proactiva de **auditoría continua** (mapped to Zero Trust principle of continuous monitoring).

- **Extender alarmas/monitorización a más eventos:** Solo creamos una alarma de login fallido, pero podríamos agregar muchas más en CloudWatch: detección de múltiples intentos de *AssumeRole* fallidos (indicando token temporal comprometido), de creación inesperada de usuarios IAM, de modificaciones a Security Groups fuera de horario, de acceso inusual a objetos S3 (p. ej. muchos GET anómalos), etc. Asimismo, habilitar **CloudTrail Insights** (si se permitiera fuera del free tier) ayudaría a detectar automáticamente patrones inusuales. Otra mejora: activar **VPC Flow Logs** y analizar tráfico de red, integrando con CloudWatch Logs para detectar, por ejemplo, intentos de escaneo de puertos en la subred (aunque con todo cerrado sería raro, pero es un plus).

- **Integración con AWS GuardDuty/Security Hub (cuando esté disponible):** En un entorno no limitado al Free Tier, sin duda se habilitaría **Amazon GuardDuty**, que aplica inteligencia de amenazas a los logs de CloudTrail, Flow Logs y DNS Logs para detectar actividad maliciosa (como uso de credenciales filtradas, comportamiento de instancias comprometidas, etc.). GuardDuty puede servir como un *PDP/PIP* externo muy potente, alertando de amenazas sutiles que nuestras alarmas manuales podrían no cubrir. Igualmente, **AWS Security Hub** podría centralizar el hallazgo de incumplimientos (ex. si algún recurso nuevo no sigue las mejores prácticas de cifrado, etc.). Estas herramientas complementarían la arquitectura Zero Trust proporcionando un monitoreo más amplio y recomendaciones de mejoras.

- **Micro-segmentación adicional con Network Firewall / WAF:** Para ambientes de producción, se podría introducir **AWS Network Firewall** o ACLs que limiten aún más el tráfico incluso dentro de la VPC (por ejemplo, inspeccionando tráfico entre bastion y BD, aunque solo es SQL, pero se podría bloquear otros protocolos). También si se expone alguna aplicación web, usar **AWS WAF** para filtrar ataques a nivel HTTP. En nuestro caso no había frontales web, pero es una expansión lógica si sumamos capas de aplicación.

- **Gestión de identidades más granular:** Actualmente, el Identity Center otorga rol admin y read-only. Podríamos crear perfiles más granulados (DevOps, Auditor, etc.) con *permission sets* personalizados, aplicando **Least Privilege** al extremo. También integrar Identity Center con una identidad corporativa (AD/SSO externo) para federar usuarios reales en lugar de locales, y aplicar políticas de sesión adaptativas (por ejemplo, usar la feature de Conditional Access Control para restringir desde qué IP u horario se puede acceder a ciertos roles).

- **Rotación y uso de credenciales IAM temporales para aplicaciones:** Si tuviéramos aplicaciones en ejecución (por ejemplo, un servidor web en la VPC pública accediendo a S3 o RDS), podríamos asignarle un **Role IAM** a la instancia EC2 con permisos mínimos, en lugar de usar claves estáticas. Ya hicimos similar con Session Manager (rol para SSM), pero esto se puede extender: todas las comunicaciones entre servicios deben hacerse con roles (via AWS STS). Esto elimina completamente las *long-lived credentials*. Adicionalmente, implementar **AWS Secrets Manager** o Parameter Store para manejar las contraseñas (por ejemplo, la password del DB) en lugar de dejarlas configuradas manualmente, logrando rotación automática de secretos.

- **Segmentación multi-cuenta u organización:** En un despliegue mayor, podríamos separar entornos (Dev/Prod) en distintas cuentas AWS unidas por AWS Organizations, aplicando políticas de control de servicio (SCPs) a nivel org para asegurar que ciertas acciones no se permitan en Prod (ej. no crear recursos públicos). Esta sería una extensión del modelo Zero Trust a nivel macro: *no confiar ni siquiera en los usuarios de la misma organización sin restricciones adicionales* y aislar blast radius por cuentas.

- **Respuesta automatizada (SOAR):** Más allá de notificar, podríamos programar respuestas automáticas a ciertos eventos. Por ejemplo, cuando la alarma de login fallido se dispare, una **AWS Lambda** suscrita al SNS podría deshabilitar la cuenta del usuario afectado preventivamente, o revocar sesiones activas (con AWS IAM *User logout* API). Otro ejemplo: si se detecta un Security Group cambiado para abrir puertos, una Lambda podría revertir el cambio y enviar alerta. Esto ya entra en *Security Orchestration and Automated Response (SOAR)*, útil para acelerar la mitigación sin intervención humana.

- **Pen-testing y simulaciones periódicas:** Una mejora operativa es programar **simulaciones de intrusión** regulares (por ejemplo, usar AWS Fault Injection Simulator o simplemente scripts Python/Boto3) que intenten violar las políticas (ej. intentar listar bucket sin TLS, intentar loguearse con credenciales incorrectas repetidas veces, escanear puertos del bastion, etc.) para comprobar que los controles responden adecuadamente y que las alertas saltan. Esto validaría continuamente la postura Zero Trust bajo distintos escenarios.

- **Mejoras de rendimiento/costos:** Desde un punto de vista más técnico, se podría habilitar **cache local** para CloudTrail (aunque es minor), consolidar logs en un data lake para análisis con Athena (CloudTrail Lake), etc. Sin embargo, estas son mejoras operativas. En Zero Trust en particular, podríamos habilitar **session timeouts más cortos** en IAM Identity Center (que el token SSO expire en menos tiempo) o requerir re-autenticación MFA para operaciones sensibles (usando *IAM MFA Context* en políticas).

- **User Experience vs Security:** Evaluar el balance. Por ejemplo, podríamos habilitar **MFA obligatoria por CLI/API** usando *Session Tags* o condiciones (actualmente la MFA la aplicamos al SSO portal; si alguien generara tokens de CLI, podríamos requerir una OTP as part of assuming role using sts:Tag). Es avanzado pero factible: así, no solo la consola pide MFA, sino también llamadas de API con roles privilegiados.

Estas mejoras llevarían la arquitectura a un nivel aún más maduro de Zero Trust, acercándola a un entorno empresarial de producción, manteniendo al mismo tiempo la filosofía de *"confiar nunca, verificar siempre, monitorear todo"*.

Riesgos Potenciales y Mitigaciones

A pesar de la cuidadosa aplicación de principios de Zero Trust, ninguna arquitectura es completamente invulnerable. A continuación se discuten algunos riesgos residuales o potenciales en este entorno y cómo podrían mitigarse:

- **Compromiso del Bastion Host:** El bastion es el único punto expuesto (aunque limitado por IP y MFA indirectamente). Si un atacante lograra acceso a la instancia bastion (por ejemplo, mediante exploit de día cero en el sistema operativo, o robo de la clave SSH del admin), podría pivotar a la red privada.

  Mitigaciones: mantener el bastion siempre actualizado (parches de seguridad), restringir aún más su acceso (idealmente *cerrar puerto 22* y usar solo Session Manager), usar autenticación multi-factor a nivel sistema (ej. OTP en login SSH también). Además, aplicar principios de *Just-in-time access*: encender el bastion solo cuando se necesite (apagado el resto del tiempo), o usar *EC2 Connect* que genera claves temporales. Monitorizar activamente el bastion (CloudWatch podría alarmar si alguien inicia sesión fuera de horario esperado). En caso de compromiso, tener listas *Network ACLs* o *SCPs* para aislarlo rápidamente. En Zero Trust, se asume brecha: por eso, incluso si bastion cae, los datos RDS están cifrados y requieren credenciales DB (que el bastion no almacena en texto plano). Aún así, bastion es un recurso crítico a proteger.

- **Robo de credenciales SSO del Admin:** Si las credenciales del administrador (Identity Center) se vieran comprometidas (phishing, keylogger), un atacante podría intentar autenticarse. Tenemos MFA que dificulta esto, pero existe la posibilidad de *MFA fatigue* (ataques donde envían muchas notificaciones push a ver si el usuario acepta por error) – en nuestro caso MFA es TOTP, así que menos vulnerable.

  Mitigaciones: concienciación del usuario admin, uso de MFA hardware (Yubikey/WebAuthn) más robustos que soft tokens, políticas de cambio periódico de contraseña admin. AWS SSO no admite aun IP whitelisting directamente, pero podríamos integrar con un IdP que sí y usarlo. Además, podríamos habilitar *AWS SSO Session Policies* (feature) para que inclusive tras login, ciertas operaciones sensibles re-pidan MFA. Adicionalmente, mantener monitorizada cualquier actividad anómala del admin (CloudTrail Insights, etc.). En Zero Trust se aboga por *principio de mínimos privilegios también para admins*: tener break-glass account aparte y usar admin principal solo para tareas mayores, usando roles de menor privilegio para lo diario.

- **Errores de Configuración (Misconfiguration):** Un riesgo común es el error humano que rompa alguna regla. Ejemplo: que alguien por error quite la bucket policy, o abra un SG a 0.0.0.0/0. Dado que somos un único admin, es controlable, pero en entornos con múltiples operadores, esto puede pasar.

  Mitigaciones: aquí ayudaría AWS Config con reglas obligatorias (ej. una regla que detecte si *Block Public Access* de S3 se desactiva, y automáticamente volver a activarla, o alerta). También las SCPs a nivel organización: podríamos poner una SCP que impida quitar *BlockPublicAcess* en S3, o que impida crear SG con 0.0.0.0/0 en ciertos puertos. Otra mitigación es el enfoque de "infraestructura como código" (CloudFormation/Terraform): si todo está codificado y versionado, es menos propenso a cambios manuales accidentales, y más fácil detectar drift. Auditorías periódicas manuales también (ej. revisar lista de SG, buckets, comparando con baseline). Zero Trust no elimina errores de config, así que hay que apoyarse en procesos y herramientas de IaC/Config checks.

- **Limitaciones del Free Tier (Cobertura parcial):** Por permanecer en Free Tier, algunos servicios no se usaron o están limitados. Por ejemplo, no tenemos GuardDuty analizando logs DNS o VPC Flow Logs, así que un ataque de tipo *exfiltración lenta de datos* (difícil en esta arq, pero supongamos via DNS tunneling) no sería detectado con nuestras alarmas simples.

  Mitigación: reconocer estas limitaciones y planificar habilitar esos servicios en cuanto sea viable. Mientras, se podría implementar alguna lambda personalizada que monitoree logs DNS o patrones de tráfico, pero es complejo. También, CloudTrail en free tier no incluye eventos Data para S3 de más de un bucket gratuito (tenemos uno, bien) pero si tuviéramos más buckets, los data events extra cuestan – hay que vigilarlos. Aquí el riesgo es perder visibilidad por restricción de costos. La mitigación es priorizar qué monitorear (ya hicimos con login, etc.) y buscar soluciones creativas (por ej, limitarnos a un bucket de logs central para todo y solo ese monitorear).

- **Persistencia de amenazas internas:** Zero Trust asume amenazas internas también. Si un empleado malicioso obtuviera, con su acceso legítimo, entrar al bastion y ejecutar comandos en la DB (que tiene credencial admin MySQL), podría filtrar datos (aunque no fuera exfiltración por red, podría copiar data y enviarla fragmentada).

  Mitigaciones: aplicar control de usuarios dentro de la DB (no dar credencial admin de MySQL a cualquiera; usar IAM DB Auth si disponible – RDS MySQL no lo soporta, pero RDS PostgreSQL sí puede integrarse con IAM roles). También segmentar duties: no todos los admins SSO necesitan acceso a todo; podríamos tener roles distintos (un dba con acceso RDS pero no a S3, etc.), así un insider no tenga llaves de todo. Y por su puesto, logging – que ya está, así podríamos forensear luego. Data Loss Prevention (DLP) no la implementamos, sería external.

- **Denegación de Servicio (DoS):** Un atacante no necesariamente busca robar datos, quizás solo derribar el servicio. Podría intentar inundar nuestra IP del bastion con tráfico (volumétrico DDoS). AWS por defecto provee mitigaciones a nivel infra (AWS Shield Standard) para IPs EC2, pero no es infalible ante ataques masivos. Igualmente, intentar saturar la DB con consultas (si obtuviera credencial DB) – RDS tiene límites y podría volverse no responsiva.

  Mitigaciones: para DDoS, se podría colocar la instancia bastion detrás de un *AWS Global Accelerator* con protección avanzado, pero no usual. Dado que bastion es para admin y no servicio público, el riesgo DDoS es bajo (no anunciado). Aun así, se puede monitorear latencia y CPU del bastion via CloudWatch y recibir alertas de uso inusual. Para la DB, usar *Performance Insights* y alertar si hay queries heavy inesperadas. Zero Trust se enfoca en seguridad acceso, no tanto performance, pero es bueno no olvidar la disponibilidad – pilar de la triada CIA – a veces puesta en riesgo por exceso de restricciones. Aquí mantuvimos disponibilidad (no bloqueamos admin fuera por error, etc., salvo try).

- **Shadow admins /Backdoors:** Un riesgo a controlar es la creación de usuarios o roles ocultos con amplios permisos (por compromiso de admin). Con CloudTrail habilitado, veríamos si alguien crea un usuario IAM y le da keys. Igualmente, la CloudTrail alarm puede ampliarse para alertar en *CreateUser* o *AttachUserPolicy* events.

  Mitigación: convertir esas en alarmas también. Además, rotación de acceso tokens SSO regulares. Zero Trust implica *no confiar en nadie, ni siquiera en administradores sin verificación*, por lo que políticas de 4-eyes (requiere aprobación 2 personas para cambios) podrían considerarse en orgs real.

Los **riesgos mayores** identificados son: compromiso de credenciales admin, compromiso de bastion, errores de configuración.

Estos se han reducido significativamente con las medidas implementadas, pero no se eliminan al 100%. La estrategia es varias capas de compensación: si falla una, otra detecta o bloquea. Mitigar riesgos es un proceso continuo: por ello la arquitectura debe evolucionar incorporando más automatización (GuardDuty, Config), revisiones periódicas y formación de usuarios. Los principios Zero Trust nos guían a siempre preguntar "*¿Qué pasa si X se compromete?*" e introducir controles para que ese escenario no se convierta en brecha catastrófica. Con las mitigaciones propuestas, elevamos aún más la resiliencia frente a las amenazas.

Conclusiones

En este proyecto se logró **diseñar e implementar una arquitectura Zero Trust funcional en AWS utilizando únicamente servicios Free Tier**, demostrando que es posible alcanzar un alto nivel de seguridad sin herramientas de costo adicional. A lo largo del informe se integraron los componentes clave – IAM Identity Center, IAM Roles/Policies, VPC, Subnets, Security Groups, EC2 Bastion, RDS, S3, KMS, CloudTrail, CloudWatch, SNS – y se describió cómo cada uno aporta a los principios de Zero Trust.

Las fases implementadas abarcaron todos los planos: **identidad**, **dispositivo/red**, **aplicación/datos** y **visibilidad/respuesta**. En particular:

- Se estableció un **Plano de Control de Identidad** robusto: autenticación federada con MFA obligatoria, eliminación de credenciales permanentes, privilegios granulares. IAM Identity Center actuó como PAP/PDP central, y se aplicó MFA para “verificación continua” en cada login. Esto asegura que solo usuarios verificados y autorizados acceden a la nube, sin confiar en contraseñas estáticas.

- Se implementó **microsegmentación de red** creando una VPC aislada con subredes segregadas. La única vía de ingreso fue un bastion fortificado (PEP) y la base de datos se mantuvo invisible al exterior (sin IP pública). Los Security Groups funcionaron como guardias de paso, permitiendo comunicaciones solo entre componentes explícitamente aprobados. Así se eliminó la confianza implícita en la red interna – cada flujo fue autorizado.

- Se reforzó la **protección de datos** cifrando todo en reposo (S3 con SSE, RDS con KMS) y en tránsito (TLS requerido) y aplicando políticas condicionales Zero Trust (denegando tráfico no seguro). Los datos en S3 y RDS quedan ilegibles sin las credenciales y claves apropiadas, reduciendo impacto incluso si hubiera acceso no autorizado a almacenamientos.

- Se habilitó la **monitoreación y trazabilidad completas** mediante CloudTrail registrando todas las acciones, CloudWatch analizando logs en tiempo real y SNS notificando incidentes. Esto cumple el principio de *"asumir brecha"* – siempre estar vigilando por si algún control falla o es evadido. La alarma de intento de login fallido demostró la capacidad de detección temprana y respuesta (al menos notificación inmediata). Con logs detallados, se pueden realizar auditorías y forenses cuando se requiera, apoyando la resiliencia.

En síntesis, el entorno construido **satisface los objetivos** planteados: se consiguió un entorno AWS endurecido que aplica *Zero Trust Architecture* en todas sus capas, usando servicios estándar. Cada acceso está autenticado con MFA, cada recurso está por defecto no confiable hacia otros (salvo reglas explícitas), y existe visibilidad de todo lo que ocurre. Se pasó de un modelo tradicional (confiar en la red corporativa, confiar en insiders) a un modelo donde **la confianza es dinámica y debe ganarse constantemente** con verificación de contexto.

Cabe resaltar que, a pesar de no usar servicios avanzados, se lograron alternativas efectivas con los recursos disponibles. Esto evidencia que **Zero Trust es más un cambio de filosofía y buenas prácticas que de tecnología específica costosa**: con configuración adecuada de IAM, red y logs, los mismos principios pueden aplicarse. Por supuesto, en un despliegue real de mayor escala se complementaría con herramientas especializadas (como AD federado, CASB, etc.), pero la base conceptual es la misma y aquí quedó demostrada.

Finalmente, el ejercicio permitió afianzar conocimientos de múltiples servicios AWS y cómo interactúan para mejorar la postura de seguridad. Se reforzó la importancia de la **configuración correcta** de cada servicio – pues un simple detalle (ej. dejar un SG abierto) podría romper la cadena Zero Trust. También se puso de manifiesto el valor de la **automatización y monitoreo** en seguridad cloud: sin CloudTrail/CloudWatch sería inviable manualmente revisar todo.

En un entorno amenazante donde las brechas suelen ocurrir por eslabones débiles, adoptar Zero Trust ayuda a minimizar daños: si un componente se ve comprometido, los demás aún resisten (ej., un bastion no da acceso a DB sin credencial; una credencial no da acceso sin MFA; etc.). Esta **defensa en profundidad** inspirada en Zero Trust eleva significativamente la resiliencia global.

Como conclusión, la arquitectura desplegada cumplió con creces las expectativas: los controles implementados funcionaron según lo planificado (validado con las evidencias recolectadas) y se logró un entorno **seguro, monitorizado y ajustado a Zero Trust** que puede servir de base para entornos productivos más complejos. El proyecto demuestra que la adopción de Zero Trust en la nube es alcanzable paso a paso, y sienta las bases para futuras mejoras y escalamiento. En adelante, se debería continuar iterando sobre este entorno, incorporando las mejoras sugeridas y adaptándose a nuevas amenazas, en línea con el espíritu de Zero Trust: *"mejorar continuamente, nunca asumir que estamos completamente seguros"*.

Referencias

- Cristian Jiménez. *AWS Zero Trust Architecture Hardening & Monitoring (Free Tier Edition)* (Subido al Github)– Documento base del laboratorio, que describe la implementación paso a paso de arquitectura Zero Trust en AWS Free Tier y cuyos contenidos han sido incorporados y ampliados en este informe.

- NIST Special Publication 800-207 (2020). *Zero Trust Architecture*[1][5] – Marco teórico del modelo Zero Trust. Define los principios fundamentales (no confianza implícita, verificación continua, proteger recursos en vez de perímetros) y componentes lógicos (PAP, PDP, PEP, PIP) empleados en el diseño de la solución.

- AWS Identity and Access Management (IAM) – Documentation. *¿Qué es IAM?*[3], AWS IAM User Guide. Explica la gestión de usuarios, roles y políticas en AWS, base para la configuración de identidades y privilegios mínimos en la fase 1.

- AWS Identity Center (antes SSO) – AWS Docs & Blog. Incluyendo uso de MFA en SSO y asignación de permission sets. Relevante para Fase 0 en establecimiento de federación con MFA.

- AWS Systems Manager – *Session Manager Beneficios* (User Guide). Describe cómo Session Manager permite acceso seguro sin puertos abiertos ni bastiones tradicionales, concepto clave usado en Fase 3 para reemplazar SSH y mejorar la postura Zero Trust.

- Amazon VPC – *¿Qué es Amazon VPC?* (AWS Docs) y *Security best practices for VPC*. Referencias sobre diseño de subredes, uso de Internet Gateways, route tables separadas y Security Groups. Aportaron guía en la implementación de la segmentación de red en Fase 3.

- AWS CloudTrail – AWS Documentation y NinjaOne Blog. *Servicio de registro de AWS CloudTrail* y *Definición de CloudTrail*. Útiles para configurar auditoría en Fase 5 y entender qué cubre (eventos de gestión, data events, etc.).

- AWS CloudWatch – AWS Docs. *¿Qué es Amazon CloudWatch?* y ejemplos de métricas y alarmas. Empleados en Fase 5 para armar las alarmas de seguridad basadas en logs (Failed logins).

- AWS SNS – AWS Docs. *¿Qué es Amazon SNS?*. Detalles del servicio de notificaciones usado en Fase 5 para alertar vía correo ante incidentes.

- AWS Well-Architected Framework – Security Pillar Whitepaper (resumen). Orientó buenas prácticas generales: habilitar MFA, principio de privilegio mínimo, logging activo, etc., todas aplicadas a lo largo del proyecto.

- Material del Curso Designing & Implementing a Zero Trust Architecture – EC-Council. Además del documento base, se tomaron lineamientos presentados en el curso respecto a Zero Trust (por ejemplo, recomendación de MFA siempre, monitoreo continuo, segmentación estricta) y se buscaron implementaciones concretas en AWS.
- Documentación oficial de cada servicio AWS mencionado (IAM, S3, VPC, RDS, KMS, CloudTrail, CloudWatch, SNS) para detalles de configuración y opciones disponibles en Free Tier.

(La mayoría de las configuraciones y comportamientos se validaron directamente en la plataforma AWS al ejecutar el laboratorio, y se citaron referencias oficiales para fundamentar las decisiones y características de seguridad implementadas.)

[1] SP 800-207, Zero Trust Architecture | CSRC

https://csrc.nist.gov/pubs/sp/800/207/final

[2] [5] Zero Trust Architecture

https://dreamlab.net/wp-content/uploads/2024/11/Arquitectura_de_confianza_cero-NIST.SP_.800-207.pdf

[3] [4] ¿Qué es IAM? - AWS Identity and Access Management

https://docs.aws.amazon.com/es_es/IAM/latest/UserGuide/introduction.html