UNIVERSIDAD MAYOR, REAL Y PONTIFICIA DE
SAN FRANCISCO XAVIER DE CHUQUISACA

VICERRECTORADO

CENTRO DE ESTUDIOS DE POSGRADO E INVESTIGACIÓN
FACULTAD DE CIENCIAS Y TECNOLOGÍA

“PRÁCTICAS DE SEGURIDAD EN INFRAESTRUCTURA COMO CÓDIGO CON TERRAFORM COMPLIANCE EN DEVOPS”

TRABAJO EN OPCIÓN A DIPLOMADO EN DEVELOPEMENT OPERATIONS "DEVOPS" V.1.

AUTOR: Mamani Chavarría Joel Tito

Sucre - Bolivia
2024

CESIÓN DE DERECHOS

Al presentar este Proyecto, como uno de los requisitos para la obtención en DevOps de la Universidad Mayor Real y Pontificia de San Francisco Xavier de Chuquisaca, autorizo al Centro de Estudios de Posgrado e Investigación y/ o a la Biblioteca de la Universidad, para que se haga de este Trabajo un documento disponible para su lectura, según las normas de la Universidad.

Asimismo, manifiesto mi acuerdo en que se utilice como material productivo, dentro del Reglamento de Ciencia y Tecnología, siempre y cuando esta utilización no suponga ganancia económica, ni potencial.

También cedo a la Universidad de San Francisco Xavier de Chuquisaca los derechos de publicación de este trabajo o parte de ella, manteniendo derechos de autor, por un periodo de treinta meses después de su aprobación.

Joel Tito Mamani Chavarría

Sucre, abril de 2024

AGRADECIMIENTOS

A Dios, porque de ti Dios proceden la riqueza y esplendor; tú gobiernas todo, en tus manos están la fuerza y el poder, y eres tú quien engrandece y fortalece a todos. Por eso, Dios nuestro, te agradezco, y a tu glorioso nombre te damos gracias y alabamos (1 Crónicas 29:12-13).

A mi madre Ferminia, padre Manuel y mis herman@s por ayudarme e incentivarme ya que sin su apoyo no se hubiera logrado este sueño.

A la Universidad Mayor Real y Pontificia de San Francisco Xavier de Chuquisaca, Facultad de Ciencias y Tecnología, por haberme cobijado en sus aulas durante mi formación académica.

A mis docentes que han compartido sus conocimientos y valores en mi formación académica.

A mis compañeros que han compartido su alegría en estos años en particular a Carlos, Cristian, Américo, Alex, Yhony y Favio.

A todos los mencionados, mis más sinceros agradecimientos.

DEDICATORIA

” Doy gracias a Dios por darme fuerza y valor para culminar esta etapa de mi vida. A mis padres por motivar e incentivarme qué ha servido de mucho.

Gracias por todo, familia este trabajo es para ustedes”

RESUMEN

En la era digital actual, la seguridad de la infraestructura de tecnología de la información (TI) es una preocupación fundamental para las organizaciones en todos los sectores. Con el aumento de las amenazas cibernéticas y los ataques sofisticados, es crucial implementar medidas efectivas para proteger los activos de información y garantizar la integridad, confidencialidad y disponibilidad de los sistemas. En este contexto, el enfoque de DevOps, que promueve la integración continua, la entrega continua y la automatización en el desarrollo de software y la gestión de la infraestructura, ha ganado una amplia adopción en las empresas. Sin embargo, este enfoque también plantea desafíos únicos en términos de seguridad, ya que la velocidad y la agilidad de los procesos de desarrollo y despliegue pueden conducir a la introducción de vulnerabilidades y riesgos en la infraestructura.

En este estudio, nos centraremos en explorar las prácticas en infraestructura como código con Terraform Compliance en entornos DevOps. Esta metodología busca garantizar la integridad y la seguridad de la infraestructura de TI al incorporar controles automatizadas en el proceso de implementación y gestión de la infraestructura.

CONTENIDO

FIGURAS

INTRODUCCIÓN

1.1 ANTECEDENTES:

En el entorno actual de desarrollo de software, la automatización y la infraestructura como código (IaC) se han convertido en prácticas comunes para facilitar la implementación y el mantenimiento de la infraestructura de TI. La adopción de herramientas de orquestación como Terraform ha permitido a las organizaciones gestionar su infraestructura de forma eficiente y escalable, mediante la definición de configuraciones como código.

Si bien la infraestructura como código ofrece numerosos beneficios en términos de agilidad y consistencia, también introduce nuevos desafíos de seguridad. Los errores en las configuraciones de infraestructura pueden tener consecuencias graves, como la exposición de datos sensibles o la interrupción del servicio. Por lo tanto, es crucial implementar prácticas de seguridad sólidas para garantizar que la infraestructura definida como código sea segura y cumpla con los estándares de seguridad establecidos.

A medida que las organizaciones adoptan la infraestructura como código, también reconocen la necesidad de integrar la seguridad en todo el proceso de desarrollo y despliegue. Las prácticas de seguridad por escaneo y monitoreo en infraestructura como código han surgido como una forma efectiva de identificar y remediar vulnerabilidades de seguridad en las configuraciones de infraestructura antes de que se implementen en entornos de producción. (AWS, 2023)

Terraform Compliance es una herramienta de código abierto diseñada para ayudar a las organizaciones a verificar y mantener la conformidad con políticas de seguridad en su infraestructura como código escrito con Terraform. Permite definir políticas de cumplimiento como código y ejecutar escaneos automáticos para identificar desviaciones de estas políticas en los archivos de configuración de Terraform (Hat, 2022).

Su integración en prácticas de seguridad y DevOps puede mejorar la seguridad y la conformidad de la infraestructura de manera proactiva.

1.2 JUSTIFICACIÓN

La investigación sobre prácticas de seguridad por escaneo en infraestructura como código con Terraform Compliance es fundamental para mejorar la seguridad de los entornos de DevOps. Esta investigación tiene como objetivo proporcionar una comprensión más profunda de cómo las organizaciones pueden integrar prácticas de seguridad efectivas en su proceso de desarrollo de infraestructura como código utilizando herramientas como Terraform Compliance. Al comprender los beneficios y desafíos asociados con esta integración, las organizaciones pueden mejorar su postura de seguridad y reducir el riesgo de incidentes de seguridad relacionados con la infraestructura.

2. SITUACIÓN PROBLEMÁTICA:

En nuestro entorno actual de desarrollo y operaciones de TI, gestionar la seguridad de la infraestructura como código es un desafío significativo.

En el contexto actual de desarrollo de software, la adopción de la infraestructura como código (IaC) ha crecido significativamente, permitiendo a las organizaciones gestionar su infraestructura de TI de forma programática y automatizada. Sin embargo, esta transición hacia la IaC ha planteado nuevos desafíos en términos de seguridad de la infraestructura. Se ha observado un aumento en el número de incidentes de seguridad relacionados con configuraciones incorrectas o inseguras en los archivos de configuración de Terraform, lo que ha llevado a vulnerabilidades en la infraestructura y la exposición de datos sensibles.

- Vulnerabilidades en las configuraciones de Terraform. Por la Complejidad de las configuraciones de infraestructura, falta de conocimiento en seguridad de la infraestructura como código. como efecto Riesgo de compromiso de seguridad, exposición de datos sensibles, interrupción del servicio.
- Falta de consistencia en la aplicación de políticas de seguridad. por Falta de procesos estandarizados, falta de herramientas para aplicar políticas de seguridad de manera uniforme. como Efecto Creación de entornos de infraestructura inseguros, incumplimiento de regulaciones y estándares de seguridad.

- ❖ Dificultades en la identificación de vulnerabilidades por la Falta de herramientas especializadas para escanear y monitorear la seguridad de la infraestructura como código. como Efecto Incapacidad para identificar y mitigar proactivamente vulnerabilidades, riesgo continuo de exposición a amenazas de seguridad.
- ❖ Complejidad en la gestión de permisos y acceso por la Falta de procesos claros para gestionar permisos y accesos en las configuraciones de Terraform. Provocando la Posibilidad de acceso no autorizado, exposición de recursos críticos, violaciones de seguridad.
- ❖ Escasa visibilidad y control sobre la seguridad de la infraestructura. por Falta de herramientas de monitoreo y auditoría para evaluar continuamente la postura de seguridad de la infraestructura como código. esto provoca dificultad para detectar y responder a amenazas, incapacidad para cumplir con requisitos de cumplimiento y regulaciones de seguridad.

La ausencia de un enfoque sistemático para escanear y monitorear la infraestructura como código deja vulnerable a posibles ataques y brechas de seguridad.

3. FORMULACIÓN DEL PROBLEMA DE INVESTIGACIÓN:

¿Cómo puede la implementación de prácticas de seguridad en infraestructura como código con Terraform Compliance en entornos DevOps mejorar la seguridad y la conformidad de la infraestructura de TI, y cuáles son los desafíos y consideraciones claves?

4. OBJETIVO GENERAL:

Evaluar el impacto de la implementación de prácticas de seguridad en infraestructura como código con Terraform Compliance en entornos DevOps.

5 OBJETIVOS ESPECÍFICOS:

1. Analizar el estado actual de la seguridad de la infraestructura como código y los desafíos asociados.

2. Investigar las características y capacidades de Terraform Compliance en la realización de escaneos de seguridad en infraestructura como código definido en Terraform.

3. Analizar un entorno para la identificación de vulnerabilidades y el cumplimiento de políticas de seguridad en la infraestructura como código.

4. Evaluar los resultados de los escaneos de seguridad con Terraform Compliance y prácticas de seguridad existentes.

5. Proponer prácticas de seguridad en infraestructura como código con Terraform Compliance en entornos DevOps, con el objetivo de mejorar la seguridad y la conformidad de la infraestructura de TI.

6. DISEÑO METODOLÓGICO

6.1. Tipo de Investigación

El tipo de investigación a realizar será exploratoria y descriptiva. La exploratoria se utilizará para comprender el contexto y los desafíos relacionados con la implementación de prácticas de seguridad en infraestructura como código. La investigación descriptiva permitirá analizar y describir los resultados obtenidos durante el estudio y proporcionará una evaluación del impacto de las prácticas de seguridad implementadas.

6.2 Métodos

Se emplearán métodos cualitativos y cuantitativos para recopilar y analizar datos. Los métodos cualitativos, como grupos focales, se utilizarán para obtener una comprensión de

las percepciones y experiencias de los profesionales de seguridad y desarrollo de software en relación con la implementación de prácticas de seguridad en infraestructura como código. Los métodos cuantitativos, como encuestas y análisis, se utilizarán para recopilar datos sobre la eficacia y el impacto de las prácticas de seguridad implementadas.

Se analizarán casos de estudio y experiencias prácticas de organizaciones que han implementado Terraform para gestionar su infraestructura. Esto permitirá identificar desafíos comunes, mejores prácticas y lecciones aprendidas en el manejo seguro de datos y la configuración de Terraform.

6.3 Técnicas

1. **Análisis Documental:** Se analizarán documentos técnicos, guías de seguridad, informes de incidentes y otras fuentes de información para recopilar datos relevantes sobre el manejo seguro de datos y la configuración de Terraform.
2. **Encuestas a Expertos:** Se llevarán a cabo encuestas con expertos en seguridad de la información y profesionales de DevOps para obtener información adicional sobre las prácticas recomendadas y los desafíos asociados con el manejo de datos confidenciales y la configuración segura de Terraform.

6.4 Instrumentos

1. **Matriz de Análisis:** Se creará una matriz de análisis para organizar y categorizar la información recopilada durante la revisión de literatura y los estudios de casos. Esto ayudará a identificar patrones y temas recurrentes en los datos.

CAPÍTULO I

MARCO TEORICO Y CONTEXTUAL

1.1 MARCO CONCEPTUAL.

1.1.1 Infraestructura como Código (IaC):

La teoría de la infraestructura como código postula que la infraestructura de TI debe tratarse y gestionarse de la misma manera que el software tradicional, utilizando código y herramientas de desarrollo para su implementación y gestión. Esta teoría sostiene que la automatización de la infraestructura a través del código facilita la implementación, el mantenimiento y la escalabilidad de los entornos de TI, al tiempo que permite una gestión más eficiente y consistente de los recursos de infraestructura (Learn, 2023).

1.1.2 DevOps:

DevOps es una cultura, filosofía y práctica que busca integrar el desarrollo de software (Dev) con las operaciones de TI (Ops) para lograr una entrega continua y confiable de software. Promueve la colaboración, la automatización y la mejora continua en todos los aspectos del ciclo de vida del desarrollo de software (NetApp, 2023).

1.1.3 Seguridad de la Infraestructura como Código:

La teoría de la seguridad de la infraestructura como código se centra en la aplicación de principios y prácticas de seguridad de la información a la gestión de la infraestructura de TI mediante código. Esta teoría aborda la importancia de implementar prácticas de seguridad desde las etapas iniciales del desarrollo de la infraestructura, utilizando herramientas y procesos de automatización para garantizar la integridad, confidencialidad y disponibilidad de los sistemas y datos (Networks, 2024).

1.1.4 Escaneo y Monitoreo Continuo de la Infraestructura:

Esta teoría sugiere que el escaneo y monitoreo continuo de la infraestructura de TI son fundamentales para identificar y mitigar riesgos de seguridad, detectar cambios no autorizados y garantizar el cumplimiento de políticas de seguridad y regulaciones. La teoría sostiene que la detección temprana de vulnerabilidades y la respuesta proactiva a las amenazas cibernéticas son críticas para mantener la integridad y la disponibilidad de los sistemas de información (Dotcom-Monitor, 2023).

1.1.5 Terraform Compliance:

Terraform Compliance es una herramienta que permite evaluar y garantizar el cumplimiento de políticas de seguridad y estándares de cumplimiento en la infraestructura definida como código mediante la implementación de reglas de auditoría y controles de seguridad (IONOS, 2019).

1.2. MARCO TEÓRICO

1.2.1 principios de la ciberseguridad

Figura 1: *Cap. 2 Seguridad de la información.*

Fuente: *https://www.grouphacking.com/wp-content/uploads/2023/07*

Confidencialidad de la información

También conocida como privacidad, hace referencia a que la información solo debe ser conocida por las personas que necesitan conocerla y que han sido autorizadas para ello. Este principio asegura que la información no va a ser divulgada de manera fortuita o intencionada (Vive-Unir, 2024).

Integridad de la información

Hace referencia a que la información que se encuentra almacenada en los dispositivos o la que se ha transmitido por cualquier canal de comunicación no ha sido manipulada por terceros de manera malintencionada. Esto garantiza que la información no será modificada por personas no autorizadas (Vive-Unir, 2024).

Disponibilidad de la información

Se refiere a que la información debe estar disponible siempre para las personas autorizadas para accederla y tratarla, y además puede recuperarse en caso de que ocurra un incidente de seguridad que cause su pérdida o corrupción. Es decir; permite que la información esté disponible cuando sea necesario (Vive-Unir, 2024).

Autenticidad

Garantiza la legitimidad de la información de la que dispone la organización o del autor de esta, es decir, que el autor del documento es el que aparece reflejado. El software debe ser capaz de comprobar que el usuario o la persona que firma un mensaje es quien dice ser, evitando así que un hacker consiga suplantar la identidad de otra persona (Vive-Unir, 2024).

2.2.2 DevOps y Seguridad.

A pesar de sus muchas ventajas, DevOps presenta nuevos riesgos y cambios culturales que crean retos de seguridad que, por lo general, no pueden afrontarse mediante las soluciones y prácticas de gestión de seguridad convencionales. Estos enfoques tradicionales suelen ser demasiado lentos, costosos o complejos para garantizar la entrega y el despliegue de

software automatizados en la nube o como contenedor (CyberArk Software, 2023). Estos retos incluyen:

Las credenciales con privilegios que se utilizan en DevOps son el blanco de ciberatacantes. Uno de los mayores desafíos de seguridad en los entornos de DevOps es la gestión del acceso con privilegios. Los procesos de DevOps requieren el uso de credenciales con privilegios de humanos y máquinas que son muy potentes y sumamente susceptibles de sufrir ciberataques (CyberArk Software, 2023).

Acceso humano: Con los procesos de alta velocidad, los profesionales de DevOps requieren acceso con privilegios en todos los entornos de desarrollo y producción.

Acceso de máquinas: Con los procesos automatizados, las máquinas y herramientas requieren privilegios (o permisos) elevados para acceder a los recursos sin intervención humana. como:

Herramientas de automatización: Ansible, Puppet.

Herramientas de CI/CD: Jenkins, Azure.

Herramientas de gestión de contenedores: Docker y Linux Containers (LXC).

Herramientas de orquestación de contenedores: Kubernetes, Red Hat OpenShift, Pivotal.

Los activos de nivel cero, como Ansible y Jenkins, tienen acceso a credenciales utilizadas por muchas otras herramientas.

Una vez que los atacantes obtienen credenciales con privilegios, pueden obtener acceso total a los procesos de DevOps, a bases de datos sensibles o incluso a toda la nube de una empresa. Los atacantes lo saben, y buscan cada vez más credenciales con privilegios, como contraseñas, claves de acceso, tokens y claves SSH, así como otros tipos de secretos como certificados, claves de cifrado y claves de API. Los atacantes pueden explotar credenciales no seguras en los entornos de DevOps, lo que puede dar lugar al criptojacking, filtraciones de datos y la destrucción de propiedad intelectual (CyberArk Software, 2023).

Los desarrolladores se centran en la velocidad, no en la seguridad. Los equipos de DevOps, centrados en producir el código más deprisa, a menudo adoptan prácticas no seguras fuera del ámbito de los equipos de seguridad. Estas prácticas pueden consistir en dejar secretos y credenciales incrustados en las aplicaciones y los archivos de configuración, reutilizar código de terceros sin un escrutinio suficiente, adoptar nuevas herramientas sin evaluarlas para detectar posibles problemas de seguridad y proteger las herramientas e infraestructura de DevOps de forma insuficiente (CyberArk Software, 2023).

Los enfoques a la gestión de secretos centrados en herramientas crean brechas de seguridad. Las herramientas de DevOps suelen tener algunas funciones integradas para proteger secretos. Sin embargo, estas funciones no facilitan la interoperabilidad o el intercambio seguro de secretos a través de herramientas, nubes y plataformas. A menudo, los equipos de DevOps usan las funciones incorporadas de sus herramientas individuales para gestionar secretos. Este enfoque puede dificultar la protección adecuada de los secretos, ya que no se pueden monitorizar y gestionar de manera coherente (CyberArk Software, 2023).

2.2.3 Terraform

Terraform es un software de iaC (Infraestructura como código) de código abierto para aprovisionar y administrar la infraestructura en la nube, desarrollada por Hashicorp (Jblanco, 2023).

Esta herramienta de codificación utiliza un lenguaje de configuración de alto nivel denominado HLC (HashiCorp Configuration Language). La estructura que utiliza puede dividirse en uno o varios archivos de configuración. Terraform es declarativo, por lo que en la configuración indicamos el estado final que queremos para la plataforma, no lo pasos para llegar a ese estado (Jblanco, 2023).

Terraform permite administrar cualquier infraestructura (Nubes públicas, nubes privadas, servicios SaaS) mediante proveedores de Terraform (Jblanco, 2023).

2.2.4 Tipos de Infraestructuras en Terraform

Terraform puede utilizarse para administrar diferentes tipos de infraestructuras, entre ellas:

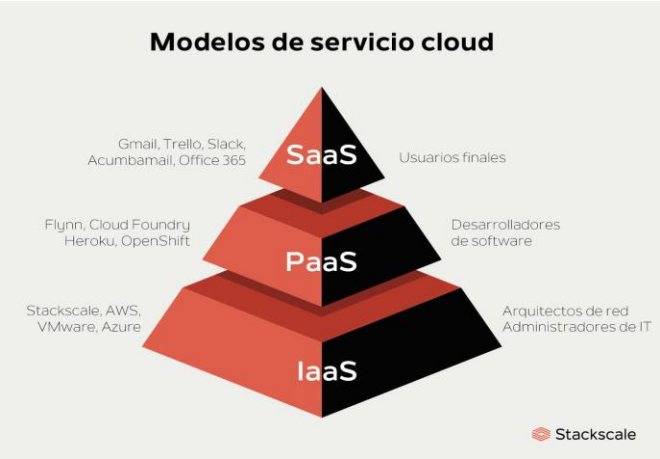
1. **Infraestructura en la nube pública:** Terraform es compatible con proveedores de nube pública como AWS, Google Cloud, Microsoft Azure, DigitalOcean, Alibaba Cloud, entre otros. Con Terraform, podemos definir, configurar y gestionar recursos de infraestructura en la nube como instancias de máquinas virtuales, redes virtuales, bases de datos, balanceadores de carga, entre otros.
2. **Infraestructura en nube privada:** Terraform también puede utilizarse para administrar infraestructuras de nube privada como OpenStack y VMware vSphere.
3. **Infraestructura de proveedor de servicios:** Terraform admite proveedores de servicios como GitHub, GitLab, Docker y Kubernetes, lo que permite la gestión de recursos como repositorios de código, imágenes de contenedores y clústeres de Kubernetes.
4. **Infraestructura local:** Terraform también se puede utilizar para gestionar recursos de infraestructura en una máquina local, como la configuración del sistema operativo o la instalación de software.

En resumen, Terraform es un gestor de infraestructura multiplataforma que te permite definir y administrar diferentes tipos de recursos de infraestructura, ya sea en la nube pública, nube privada, servicios en línea o localmente en una máquina.

2.2.5 Diferencias entre IaaS, PaaS y SaaS

SaaS, PaaS e IaaS son tres modelos de servicio diferentes dentro del campo de la computación en la nube, que ofrecen diferentes niveles de abstracción y responsabilidad a los usuarios. Las principales diferencias entre SaaS, PaaS e IaaS son las siguientes:

Figura 2: *Cap. 2.2 Modelos de servicios de la Nube*



Fuente: *https://www.stackscale.com/es/blog/modelos-de-servicio-cloud/*

- **Software como Servicio (SaaS):** SaaS es un modelo de servicio en la nube en el cual los usuarios acceden a aplicaciones de software a través de Internet, y el software es gestionado y mantenido por el proveedor de servicios en la nube. Los usuarios no tienen que preocuparse por la infraestructura subyacente, ya que todo el software, la gestión de datos y el mantenimiento son responsabilidad del proveedor. Ejemplos de SaaS son aplicaciones de correo electrónico en la nube como Gmail o plataformas de gestión de relaciones con los clientes (CRM) como Salesforce (Technologies, n.d.).
- **Plataforma como Servicio (PaaS):** PaaS es un modelo de servicio en la nube que proporciona a los usuarios una plataforma completa de desarrollo y despliegue de aplicaciones, sin tener que preocuparse por la infraestructura subyacente. Los usuarios pueden desarrollar, ejecutar y gestionar sus propias aplicaciones en la nube utilizando herramientas y servicios proporcionados por el proveedor de PaaS. Esto incluye herramientas de desarrollo, bases de datos, entornos de ejecución y servicios de escalado automático. Ejemplos de PaaS son Azure App Service de Microsoft o Google App Engine (Technologies, n.d.).
- **Infraestructura como Servicio (IaaS):** IaaS es un modelo de servicio en la nube que ofrece a los usuarios recursos de infraestructura virtual, como máquinas virtuales, redes y almacenamiento, a través de Internet. Los usuarios tienen un

mayor control sobre la infraestructura, ya que pueden configurar y gestionar los recursos según sus necesidades. Sin embargo, los usuarios también son responsables de gestionar el sistema operativo, las actualizaciones de software y otras tareas de mantenimiento. Ejemplos de IaaS son Amazon Web Services (AWS), Microsoft Azure o Google Cloud Platform (Technologies, n.d.).

2.2.6 Mejores prácticas y consideraciones de seguridad

A continuación, se presentan algunas mejores prácticas para el uso seguro y eficiente de Terraform en Azure:

1. **Gestión de credenciales:** Es importante gestionar las credenciales de Azure de forma segura y cuidadosa para garantizar que los recursos se configuren correctamente y que los datos de la cuenta de Azure estén protegidos. Se recomienda utilizar Azure Active Directory (AD) para autenticar a los usuarios y controlar el acceso a los recursos. Las credenciales de la cuenta de Azure deben mantenerse en secreto y no deben ser compartidas públicamente o almacenadas en texto plano. Terraform admite la autenticación con Azure AD a través de la integración con Azure CLI, lo que permite a los usuarios autenticarse de forma segura sin tener que almacenar credenciales de forma explícita (TerryLanfear, 2024).
2. **Seguridad de la configuración de Terraform:** Terraform utiliza archivos de configuración para definir la infraestructura, por lo que es importante asegurarse de que estos archivos sean seguros. Se recomienda utilizar un sistema de control de versiones para el código de Terraform y evitar el almacenamiento de credenciales o información confidencial en los archivos de configuración. Además, se pueden utilizar herramientas de análisis de código para identificar posibles vulnerabilidades y errores en el código (TerryLanfear, 2024).
3. **Revisión del código de Terraform:** Antes de implementar cualquier cambio en la infraestructura de Azure, es importante revisar cuidadosamente el código de Terraform para identificar posibles problemas. Esto puede incluir la revisión de los cambios propuestos en un entorno de desarrollo antes de implementarlos en producción. Además, se recomienda establecer un proceso de revisión de código en

el que otros miembros del equipo revisen y aprueben los cambios antes de la implementación (TerryLanfear, 2024).

4. **Mejores prácticas para la gestión de cambios y versiones:** Es importante gestionar los cambios y versiones de la infraestructura de forma cuidadosa y controlada. Se recomienda utilizar un sistema de control de versiones para el código de Terraform y mantener un registro de todos los cambios realizados en la infraestructura. Además, se pueden utilizar herramientas de automatización y pruebas para asegurarse de que los cambios se implementen de manera consistente y sin errores (TerryLanfear, 2024).

1.3 MARCO CONTEXTUAL.

1.3.1 Contexto Tecnológico:

La adopción de la infraestructura como código (IaC) ha revolucionado la forma en que se gestionan los entornos de TI en la era digital. Mediante el uso de código de programación, las organizaciones pueden definir y gestionar recursos de infraestructura de manera eficiente y escalable. Terraform se destaca como una herramienta líder en este ámbito, permitiendo a los equipos crear, modificar y mantener la infraestructura en la nube de forma declarativa. Esta capacidad de automatizar la infraestructura es fundamental en entornos DevOps, donde se busca integrar el desarrollo y la operación de software para lograr una entrega continua y confiable de aplicaciones (IONOS, 2019).

1.3.2 Contexto de Seguridad en DevOps:

A medida que las organizaciones adoptan prácticas DevOps e IaC, surgen nuevos desafíos y riesgos de seguridad. La automatización y la evolución de la infraestructura pueden dejar vulnerabilidades expuestas si no se aplican medidas de seguridad adecuadas. Los equipos de seguridad enfrentan desafíos como la gestión de accesos privilegiados, la detección y mitigación de configuraciones inseguras, y la protección contra amenazas persistentes. Es

esencial implementar prácticas de seguridad sólidas que aborden estos riesgos y garanticen la integridad y confidencialidad de los datos en entornos DevOps (Trevino, 2024).

1.3.3 Marco Normativo y de Cumplimiento:

Las regulaciones y estándares de cumplimiento, como GDPR, HIPAA, PCI DSS y SOC 2, imponen requisitos estrictos sobre la protección de datos y la seguridad de la infraestructura. Las organizaciones que manejan datos sensibles deben cumplir con estos estándares para evitar sanciones legales y proteger la confianza del cliente. La implementación de prácticas de seguridad en la infraestructura como código es fundamental para cumplir con estos requisitos y garantizar la seguridad y privacidad de los datos en entornos DevOps (IONOS, 2019)..

La seguridad de la infraestructura y el software es crítica para las organizaciones en todos los sectores. La seguridad puede tener graves consecuencias, incluida la pérdida de datos, la interrupción del negocio y el daño a la reputación de la marca. Por lo tanto, la seguridad se ha convertido en una prioridad estratégica para las empresas que buscan proteger sus activos digitales y mantener la confianza del cliente. La adopción de prácticas de seguridad efectivas en entornos DevOps puede mejorar la resiliencia del negocio, fortalecer la confianza del cliente y proporcionar una ventaja competitiva en el mercado en constante evolución (IONOS, 2019)..

CAPITULO II

DIAGNÓSTICO

2.1 INTRODUCCIÓN

Para el diagnóstico de la seguridad en la infraestructura como código utilizando Terraform Compliance en DevOps, se utilizó los siguientes instrumentos de recolección de datos:

2.1.1 DATOS CONFIDENCIALES Y CREDENCIALES.

Terraform necesita credenciales autorizadas para realizar acciones de API necesarias para desplegar la infraestructura definida en su código. Estas credenciales otorgan acceso privilegiado para crear, gestionar y eliminar entornos, por lo que es crucial protegerlas contra accesos no autorizados.

Para interactuar con sistemas externos y API en la nube, Terraform utiliza proveedores. Hay varios métodos para proporcionar credenciales a estos proveedores, como la especificación en la configuración del proveedor dentro del código HCL, el uso de variables de entorno y archivos de configuración. Se desaconseja encarecidamente el uso de credenciales estáticas codificadas en el código, ya que pueden ser comprometidas y expuestas a personas no autorizadas a través de sistemas de control de versiones. Es posible realizar análisis de código estático para identificar credenciales codificadas con patrones conocidos.

Proveedor de Terraform con credenciales codificadas

Figura 3: *Cap. 2.2 Proveedor de Terraform*

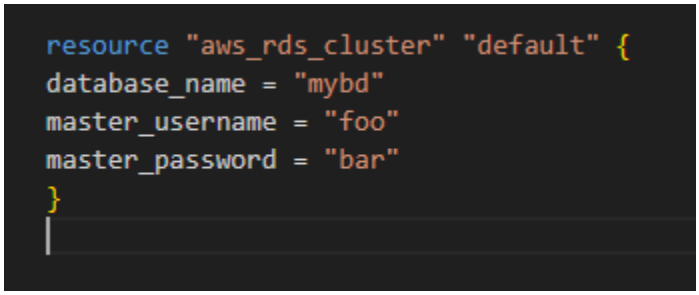
```
1  provider "aws" {
2  region = "us-cast-2"
3  access_key = "ACCESSKEY"
4  secret_key = "SECRETTKEY"
5  }
6
```

Fuente: *Elaboración propia.*

Fuera del ámbito de los proveedores, existen recursos que demandan credenciales o claves de API para su establecimiento y administración. Se debe proceder con precaución durante la configuración de estos elementos, asegurando que las credenciales no sean expuestas inadvertidamente en la salida de la Interfaz de Línea de Comandos (CLI) de Terraform. Un ejemplo de estos recursos son las bases de datos, donde al crear una nueva instancia, se requiere una credencial administrativa imprescindible para su acceso

Recurso de Terraform con credenciales codificadas.

Figura 4: *Cap. 2.2 Recursos de Terraform*



Fuente: *Elaboración propia.*

Una estrategia recomendada para evitar la inclusión de credenciales en el código fuente es suministrarlas mediante el uso de variables. Aunque este enfoque impide que las credenciales se almacenen directamente en el código, es importante tener en cuenta que los valores de las variables podrían ser visibles en el resultado o los registros de Terraform. Desde la versión 0.14 de Terraform, se ha introducido la capacidad de configurar las variables como confidenciales, lo que impide que Terraform las muestre.

Terraform con variables marcadas como confidenciales

Figura 5: *Cap. 2.2 Terraform con variables marcadas como confidenciales*

```
resource "aws_rds_cluster" "default" {
  database_name          = "mydb"
  master_username        = var.username
  master_password        = var.password
}

variable "username" {
  description = "BD Admin username"
  type        = string
  sensitive   = true
}

variable "password" {
  description = "BD Admin password"
  type        = string
  sensitive   = true
}
```

Fuente: *Elaboración propia.*

Dado que las variables que contienen el nombre de usuario y la contraseña están designadas como confidenciales, su valor no será visible en la salida de la línea de comandos al ejecutar los comandos "terraform plan" y "terraform apply"

Resultado confidencial de Terraform

Figura 6: *Cap. 2.2 Ejemplo de resultado confidencial de Terraform*

```
output "credentials" {
  value     = "${var.username}:${var.password}"
  sensitive = true
}
```

Fuente: *Elaboración propia.*

Otro posible lugar donde los secretos podrían ser almacenados es en los archivos de estado. Terraform utiliza estos archivos para rastrear la configuración de los recursos tal como se establecen en el entorno después de ejecutar un comando "terraform apply". Aunque los valores de las variables marcadas como confidenciales no se muestran en la salida de Terraform, estos valores se almacenan en los archivos de estado porque Terraform necesita mantener un registro de si el estado de estos valores ha cambiado para ejecutar correctamente las API que afectan a estos recursos. Para evitar la exposición accidental, es fundamental utilizar un backend de estado remoto de Terraform que cifre estos archivos de estado y limite estrictamente el acceso a ellos. Cuando se utiliza un backend remoto, Terraform solicita el archivo de estado del backend según sea necesario y lo mantiene en la memoria solo durante sus operaciones, lo que ayuda a evitar el almacenamiento de datos confidenciales en el disco del sistema donde se ejecuta Terraform. En la sección "Colaboración segura" de este documento, profundizaremos en los archivos de estado y cómo protegerlos.

2.1.2 ESTADO

Terraform utiliza un archivo de estado para seguir el estado de los recursos en la infraestructura suministrada. Por defecto, este archivo se guarda localmente en el sistema de archivos donde se ejecuta Terraform. Aunque esto facilita el inicio de uso de Terraform como usuario único, colaborar con otros es difícil a menos que se comparta el archivo de estado. Almacenar los archivos de estado con el código fuente es arriesgado, ya que podrían contener secretos u otra información confidencial. Para una colaboración segura, se recomienda utilizar un backend remoto de estado para almacenar y compartir los archivos de estado.

Al utilizar un backend remoto de estado, en lugar de mantenerlo localmente, Terraform recuperará el estado desde una ubicación remota, como Consul de HashiCorp, AWS S3 o el almacenamiento de blobs de Azure. El estado se mantiene en la memoria para evitar que los datos confidenciales se almacenen en el disco. Después de que las operaciones que actualizan el archivo de estado se completan, este se carga en el backend. Esto garantiza que el archivo de estado siempre contenga la versión más reciente de los recursos

suministrados por Terraform, permitiendo a los colaboradores acceder a él. Además, los backends de estado deben configurarse con un sólido control de acceso para proteger los datos en los archivos de estado, y pueden admitir cifrado en reposo y en tránsito para mayor seguridad.

.

Backend de estado de Terraform S3

Figura 7: *Cap. 2.3 Ejemplo de backend de estado de Terraform S3*

Fuente: *Elaboración propia.*

Si bien utilizar backends remotos de estado facilita la colaboración en la infraestructura, es crucial evitar realizar cambios simultáneos, ya que esto puede resultar en inconsistencias y corrupción del archivo de estado. Para prevenir esta situación, Terraform ofrece el bloqueo de estado, que impide que múltiples personas realicen cambios simultáneos en la infraestructura. Cuando se activa el bloqueo de estado, Terraform verifica si hay bloqueos antes de ejecutar cambios. Si no hay bloqueos activos, Terraform establece un bloqueo mientras aplica los cambios y lo libera una vez que el archivo de estado se actualiza con los últimos cambios. Por ejemplo, al configurar el backend de estado S3, se puede habilitar el bloqueo de estado utilizando el parámetro 'dynamodb_table' para vincularlo con una tabla DynamoDB.

Backend de estado de Terraform S3 usando DynamoDB para bloqueo de estado

Figura 8: *Cap. 2.3 Backend de estado de Terraform S3 usando DynamoDB*

```
terraform {
  backend "s3" {
    bucket = "state-bucket"
    key    = "example.tfstate"
    region = "us-east-1"
    encrypt = true
    dynamodb_table = "state-locking-table" // Habilita el bloqueo de estado utilizando DynamoDB
  }
}
```

Fuente: *Elaboración propia.*

Si bien los backends de estado y los bloqueos son útiles para colaborar de manera segura dentro de la infraestructura, es fundamental asegurarse de que el acceso a estos componentes esté estrictamente controlado. Cada backend tiene su propio mecanismo para configurar este acceso, por lo que es necesario investigar y seguir las prácticas recomendadas sobre control de acceso específicas para cada caso.

2.1.3 GESTIÓN DE MÓDULOS Y PROVEEDORES DE TERRAFORM

Terraform utiliza complementos conocidos como 'proveedores' para interactuar con las API remotas y gestionar los recursos definidos en el código. Estos proveedores se descargan localmente en el sistema donde se ejecuta Terraform durante el comando 'terraform init'. Debido a que los proveedores son responsables de operaciones críticas en la infraestructura, es fundamental descargarlos de fuentes confiables y verificar su integridad antes de usarlos.

Hasta la versión 0.14 de Terraform, el comando 'terraform init' crea un archivo de bloqueo de dependencias (.terraform.lock.hcl) que registra las versiones y los hash de los proveedores. Terraform realiza una verificación de hash para garantizar que los proveedores no hayan sido alterados después de su descarga inicial. Además, Terraform proporciona checksums firmados para los proveedores confiables, verificando su integridad

antes de su uso. Para versiones anteriores a la 0.14, se puede completar manualmente el archivo terraform.lock.hcl con los checksums necesarios para garantizar la integridad de los proveedores. Esta verificación se realiza fuera de banda.

Una forma adicional de proteger el entorno de Terraform es espejar los proveedores autorizados localmente. Terraform permite la descarga de proveedores desde sistemas de archivos locales, redes internas o repositorios privados. Al usar este método, se reduce el tráfico de red y se optimiza el rendimiento de Terraform, al tiempo que se limita el uso de proveedores a aquellos que cumplen con los requisitos de seguridad y licencia de la organización.

Los archivos de configuración de Terraform también pueden definir dependencias externas conocidas como módulos. Estos módulos permiten encapsular la configuración de Terraform en paquetes reutilizables, evitando así la duplicación de código y facilitando la implementación de prácticas recomendadas en todos los proyectos. Sin embargo, al igual que con los proveedores, es esencial garantizar la integridad de los módulos.

Los módulos se descargan localmente durante la ejecución del comando 'terraform init', y existen varias formas de especificar su ubicación, incluyendo registros de Terraform, repositorios de GitHub, rutas locales y URLs de HTTP, entre otros. A diferencia de los proveedores, los módulos no se registran en un archivo de bloqueo de dependencias, por lo que su integridad debe ser verificada manualmente antes de su uso. Durante la recuperación de módulos de repositorios remotos, Terraform no realiza ninguna verificación de integridad, lo que podría suponer un riesgo de seguridad si los artefactos versionados en los repositorios cambian su contenido de manera inadvertida.

Módulo de Terraform usando el registro de Terraform

Figura 9: *Cap. 2.4 Configuración del módulo de Terraform usando el registro de Terraform*

```
module "proyect-factory" {
  source = "terraform-google-modules/project-factory/google"
  version = "10.0.1"
}
```

Fuente: *Elaboración propia.*

Una estrategia para asegurar la integridad de los módulos empleados en tu entorno es generar checksums y crear un espejo local para los módulos públicos utilizados. Al utilizar módulos en tu código, es fundamental garantizar que solo se emplee el espejo local una vez que se ha verificado que los módulos cumplen con los requisitos de seguridad y las normativas de licencia de código abierto.

2.1.4 GESTIÓN DE INCONSISTENCIAS ENTRE EL CÓDIGO Y LA NUBE

En ocasiones, a pesar de que la infraestructura se haya creado inicialmente mediante la automatización y la Infraestructura como Código (IaC), pueden surgir cambios manuales durante la ejecución, ya sea a través de mecanismos de "romper el cristal" u otras vías. Estos cambios generan discrepancias entre el estado real del entorno y lo definido en el código de Terraform.

Una ventaja de utilizar Terraform es su capacidad para mantener un registro detallado de todos los recursos suministrados y el estado de su configuración. Dado que el código de Terraform es la representación objetiva del estado deseado de la infraestructura, cada vez que se ejecuta un comando "terraform plan", Terraform detecta las diferencias entre los recursos que están actualmente en el estado y aquellos definidos en el código como parte del plan de Terraform.

Es esencial establecer un proceso para identificar y manejar las discrepancias, ya que los requisitos de seguridad operativa codificados en la Infraestructura como Código pueden pasarse por alto cuando se realizan cambios fuera del entorno de Terraform. Estas discrepancias pueden ser administradas mediante la revisión regular de los recursos que están siendo rastreados en el estado de Terraform mediante la ejecución periódica del comando "terraform plan". Además, es importante realizar auditorías regulares del entorno para identificar recursos que no fueron creados a través de Terraform. En caso de encontrar tales recursos, Terraform ofrece la capacidad de integrarlos mediante el comando "import", el cual incorporará el estado del recurso externo al archivo de estado de Terraform, aunque se deberá proporcionar el código correspondiente en el lenguaje de configuración de Terraform (HCL).

Sin embargo, aunque estos métodos pueden ser efectivos para la detección y gestión de discrepancias, pueden no ser escalables en entornos de desarrollo de alta velocidad. Por lo tanto, se recomienda el uso de herramientas automatizadas para identificar y corregir discrepancias de manera eficiente, especialmente en entornos que requieren la gestión de discrepancias a gran escala.

2.1.5 PRÁCTICAS RECOMENDADAS DE SEGURIDAD

A pesar de que herramientas como Terraform simplifican la administración de la infraestructura, los errores de configuración que resultan en brechas de seguridad son comunes en entornos de nube. Sin embargo, con el enfoque adecuado, Terraform puede ser utilizado para implementar prácticas recomendadas en términos operativos y de seguridad.

Los módulos de Terraform son una forma efectiva de compartir y reutilizar las mejores prácticas. Estos módulos permiten codificar los requisitos operativos y de seguridad necesarios para la provisión de recursos en un entorno. Posteriormente, estos módulos pueden ser aplicados a través de pipelines de CI/CD u otros métodos de aprovisionamiento, asegurando así el uso de módulos que han sido previamente examinados.

Una herramienta valiosa para aplicar estas prácticas recomendadas es Terrascan, un analizador de código estático que evalúa los errores de configuración utilizando políticas

definidas en el lenguaje Rego y el motor OPA. Estas políticas pueden ser personalizadas para cumplir con los estándares específicos de un entorno. Al integrar Terrascan en su pipeline de CI/CD, puede escanear los cambios en sus archivos de configuración de Terraform en busca de problemas de seguridad. Si se detecta algún problema, el proceso de CI/CD fallará y proporcionará un mensaje de error indicando que se encontró una vulnerabilidad que necesita ser corregida.

Terrascan en pipelines de CI/CD

Otro enfoque efectivo para abordar los problemas de seguridad de manera eficiente es la práctica de Remediation as Code (RaC). Con RaC, las correcciones para los problemas de seguridad detectados se implementan a través de solicitudes de extracción en el sistema de control de versiones. Esto permite revisar las correcciones propuestas y fusionarlas en el código base para resolver los problemas rápidamente.

Figura 10: *Cap. 2.6 Terrascan en pipelines de CI/CD*

Fuente: *Elaboración propia*

2.1.6 MODELADO DE AMENAZAS

El modelado de amenazas es un proceso esencial en el cual se identifican y categorizan las posibles amenazas a los sistemas según el riesgo que representan. Al comprender las amenazas en el entorno, se pueden identificar y planificar las medidas necesarias para mitigarlas antes de que los agentes maliciosos las exploren. Un aspecto fundamental del

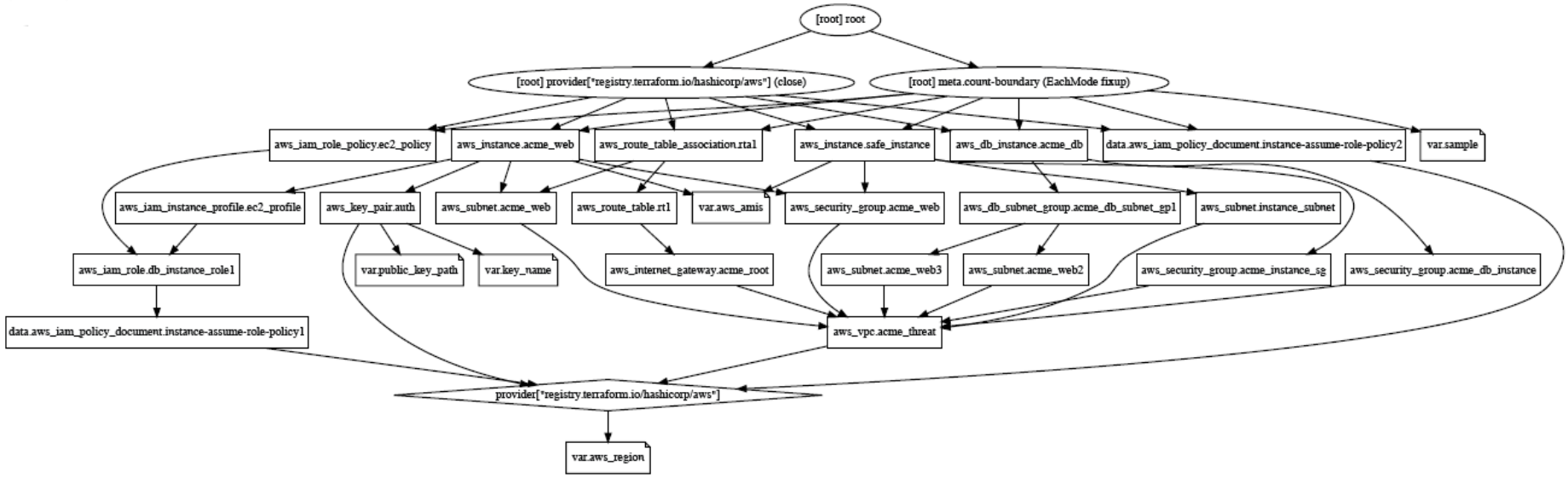
modelado de amenazas es la identificación de los recursos en la arquitectura, su configuración y las relaciones entre ellos. Esta información se utiliza para establecer los límites de confianza en el sistema y para identificar posibles vías de ataque.

Infrastructure as Code (IaC) puede ser analizado para identificar recursos, redes y relaciones de identidad entre los componentes. Terraform aprovecha estas relaciones al construir un grafo de dependencias como parte de su proceso de planificación de la infraestructura. Una forma de visualizar estas dependencias es mediante el comando 'terraform graph'

Gráfico de dependencias de Terraform

El análisis de las relaciones en Terraform puede ayudar a establecer límites de confianza en el sistema. Por ejemplo, al examinar el código de Terraform de un sistema específico, podemos identificar los límites de red definidos por grupos de seguridad, listas de control de acceso (ACL) de red y tablas de rutas. Esta información nos permite determinar si hay rutas de red que podrían exponer el sistema al público o si el sistema está aislado de Internet y solo es accesible desde otros sistemas internos. De manera similar, los límites de identidad se pueden establecer para controlar el nivel de acceso que tienen los recursos a otros componentes y sistemas. Al identificar posibles amenazas en la arquitectura, el mapa de relaciones puede ayudar a detectar posibles rutas de infiltración, donde las vulnerabilidades de seguridad en los componentes de la infraestructura podrían ser explotadas para violar los límites de confianza del sistema.

Figura 11: *Cap. 2.7 Gráfico de dependencias de Terraform*



Fuente: *Elaboración propia*

2.2 CONCLUSIONES Y RECOMENDACIONES

2.2.1 CONCLUSIONES

En conclusión, la seguridad de la infraestructura como código es un componente crítico de la estrategia de seguridad de cualquier organización moderna. Los hallazgos de este proyecto destacan la importancia de implementar un enfoque proactivo y multifacético para garantizar la integridad y protección de los recursos de infraestructura en entornos de desarrollo y producción.

Terraform es una herramienta poderosa que facilita la gestión de entornos complejos a escala. Existen múltiples consideraciones de seguridad cuando se utiliza Terraform para garantizar que su entorno no sufra riesgos. Al mismo tiempo, Terraform se puede utilizar como una herramienta para detectar las desviaciones que hayan tenido lugar en los recursos gestionados por código y para imponer el cumplimiento con sus requisitos y políticas de seguridad.

Al seguir las recomendaciones propuestas y adoptar las mejores prácticas de seguridad, la organización estará mejor preparada para enfrentar los desafíos emergentes en el panorama de amenazas actual. La inversión en herramientas especializadas, capacitación del personal y procesos de mejora continua ayudará a mitigar los riesgos de seguridad y fortalecer la postura de seguridad general de la organización en el ámbito de la infraestructura como código.

2.2.2 RECOMENDACIONES

Se recomienda establecer políticas de seguridad claras y estrictas que definan los requisitos mínimos de seguridad para las configuraciones de infraestructura como código. Esto puede incluir la aplicación de principios de mínimo privilegio, cifrado de datos sensibles y autenticación multifactor.

Mantener actualizado las políticas de seguridad en el manejo de infraestructura como código para adaptarse a los cambios de las nuevas amenazas de seguridad.

Es importante establecer un sistema de monitoreo continuo de la seguridad de la infraestructura, utilizando herramientas como Prometheus y Grafana. Esto permitirá detectar y responder rápidamente a posibles amenazas o brechas de seguridad en tiempo real.

Capacitar regularmente al equipo de desarrollo y operaciones sobre las mejores prácticas de seguridad en el uso de Terraform y otras herramientas de infraestructura como código.

Realizar auditorías periódicas de seguridad del código de Terraform y del estado de la infraestructura para identificar posibles vulnerabilidades.

Fomentar una cultura de seguridad en toda la organización, promoviendo la colaboración y la comunicación entre los equipos de desarrollo, operaciones y seguridad de la información.

BIBLIOGRAFÍA

AWS. (2023). Amazon Web Services, Inc. https://aws.amazon.com/es/what-is/iac/.

CyberArk Software. (17 de abril de 2023). *Qué es la seguridad de DevOps*. Obtenido de CyberArk: https://www.cyberark.com/es/what-is/devops-security/

Dotcom-Monitor. (2023). Escaneo y Monitoreo Continuo de la Infraestructura:. https://www.dotcom-monitor.com/es/aprende-con-dotcom-monitor/que-es-el-monitoreo-de-infraestructura/.

Hat, H. (2022). Infrastructure as Code. https://www.redhat.com/es/topics/automation/what-is-infrastructure-as-code-iac.

IONOS. (21 de Junio de 2019). Obtenido de Terraform: https://www.ionos.es/digitalguide/servidores/herramientas/que-es-terraform/

Jblanco. (5 de Septiembre de 2023). Obtenido de Terraform: qué es y beneficios que aporta para el cloud de tu empresa: https://www.plainconcepts.com/es/terraform/

Learn, M. (2023). nfraestructura como código (IAC). https://learn.microsoft.com/es-es/devops/deliver/what-is-infrastructure-as-code.

NetApp. (25 de Agosto de 2023). Obtenido de Qué es DevOps y para qué sirve?: https://www.netapp.com/es/devops-solutions/what-is-devops/

Networks, P. A. (2024). Seguridad de la infraestructura como código (IaC). https://www.paloaltonetworks.es/prisma/cloud/infrastructure-as-code-security.

Technologies. (s.f.). Obtenido de IAAS. PAAS. SAAS: https://www.veritas.com/es/mx/information-center/iaas-paas-saas

TerryLanfear. (25 de Enero de 2024). Obtenido de Características de seguridad de Azure: https://learn.microsoft.com/es-es/azure/security/fundamentals/identity-management-overview

Trevino. (19 de Abril de 2024). Obtenido de Desafíos y prácticas recomendadas de seguridad en DevOps: https://www.keepersecurity.com/blog/es/2024/04/11/devops-security-challenges-and-best-practices/

Vive-Unir. (24 de enero de 2024). *Principios de la seguridad informática*. Obtenido de UNIR: https://www.unir.net/ingenieria/revista/principios-seguridad-informatica/

ANEXOS

Anexo A. Archivo de configuración de terraform

```
terraform {
  backend "s3" {
    bucket = "state-bucket"
    key    = "example.tfstate"
    region = "us-east-1"
    encrypt = true
    dynamodb_table = "state-locking-table" // Habilita el bloqueo de estado utilizando DynamoDB
  }
}

provider "aws" {
  region     = "us-east-2"
  access_key = "ACCESSKEY"
  secret_key = "SECRETTKEY"
}

resource "aws_rds_cluster" "default" {
  database_name         = "mydb"
  master_username       = var.username
  master_password       = var.password
```

```
}

variable "username" {
  description = "BD Admin username"
  type        = string
  sensitive   = true
}

variable "password" {
  description = "BD Admin password"
  type        = string
  sensitive   = true
}

output "credentials" {
  value     = "${var.username}:${var.password}"
  sensitive = true
}

module "project-factory" {
  source  = "terraform-google-modules/project-factory/google"
  version = "10.0.1"
}
```