Framework de implementación de entornos seguros en la nube con Kubernetes y AWS

Abel Alejandro Nieva Carrizo

Máster universitario de Ciberseguridad y Privacidad

Seguridad empresarial

Profesores

Miguel Ángel Flores Terrón

Victor Garcia Font

13/06/2023

Universitat Oberta de Catalunya

Universitat Oberta de Catalunya uoc.edu

FICHA DEL TRABAJO FINAL

Título	Framework de implementación de entornos seguros en la nube con Kubernetes y AWS
Nombre del autor:	*Abel Alejandro Nieva Carrizo*
Nombre del consultor/a:	*Miguel Flores Terron*
Nombre del PRA:	*Victor Garcia Font*
Fecha de entrega (mm/aaaa):	*06/2023*
Titulación o programa:	Máster universitario de Ciberseguridad y Privacidad
Área del Trabajo Final:	*Seguridad Empresarial*
Idioma del trabajo:	*Castellano*
Palabras clave	*Seguridad empresarial, kubernetes, Amazon Web Services, Elastic Kubernetes Services, auditoría , infraestructura como código. DevSecOps, Devops , ArgoCD*

Resumen del Trabajo

El trabajo final de Máster tiene como fin el desarrollo de un framework, que permita la creación de entornos nativos en la nube dotados de herramientas y buenas prácticas de seguridad. La infraestructura como código y repositorios de código fuente, nos permiten el despliegue de clusters Kubernetes (EKS) y aplicaciones del negocio, en la nube pública de AWS, de manera segura. El trabajo consta de diferentes secciones, inicialmente se describe el proyecto y sus objetivos. Posteriormente se analizaron los conceptos teóricos en los que se fundamenta el presente trabajo. Las contramedidas o acciones de seguridad incluidas en el producto serán clasificadas de acuerdo al modelo de defensa en profundidad (seguridad nativa en la nube) propuesto por los desarrolladores de kubernetes. Finalmente se describen casos de usos de la solución desarrollada.

Abstract

This academic paper aims at developing a software framework which allows to build cloud native environments with security best practices and tools. The infrastructure as code and source code repositories enable us to deploy kubernetes clusters (EKS) and applications in AWS public cloud service in a secure manner. This paper has been splitted in chapters. At the very beginning it describes the project details and goals. Then theory concepts have been reviewed which support this paper. The actions ,strategies to remediate the issue or mitigate the risk are classified based on the security depth defense model (cloud native security) presented by kubernetes´ developers. Lastly a use case would be developed based on the proposed solution.

Índice

Lista de Imágenes

Lista de tablas

1. Introducción

1.1. **Contexto y justificación del Trabajo**

El mundo se encuentra cada vez más digitalizado. Las empresas necesitan de las tecnologías de la información y la comunicación para competir en la economía moderna y global. Sin embargo, cada nueva tecnología nos enfrenta a nuevas amenazas y vulnerabilidades. Las amenazas y los desafíos que enfrentamos hoy en día, son los más grandes de la historia de las tecnologías de la información [1]

El ciberdelito es un negocio muy rentable con una relativa baja exposición por parte del ciberdelincuente. Los cibercriminales conocen muy bien, que las personas y las empresas tienen una dependencia mayor de las tecnologías de la información, y cada día generan más y más información digital de manera consciente o inconsciente. En muchos casos, los dueños de la información, no conocen el valor de la misma, hasta que la misma se ve comprometida. La afirmación anterior afecta tanto a usuarios individuales, como a empresas.

Medir el costo de la pérdida de información es una actividad muy difícil de realizar de manera exacta. Las personas físicas no suelen calcular ni medir los costos relacionados a una brecha de seguridad, ya que no están obligadas a hacerlo y no la esperan. Pero las empresas están obligadas. Por ejemplo, podemos destacar las regulaciones gubernamentales de los diferentes países o bloques económicos como la UE, o los costos en la pérdida de imagen positiva por parte de sus clientes. Aunque sea imposible medir exactamente el costo de una incidencia de seguridad, podemos acercarnos bastante, y tomar medidas en consecuencia para evitarla.

Las empresas nativas de la nube, como las que poco a poco migran sus aplicaciones, coinciden que la arquitectura de micro-servicios es ideal para entornos, en el que la escalabilidad está garantizada. Por ello los esquemas monolíticos no aprovechan las ventajas de las nubes y ya no es una opción a tener en cuenta.

Las aplicaciones nativas de la nube son un conjunto de pequeños, independientes, y muy bajo acoplado servicios computacionales. [2]. El desarrollo de aplicaciones nativas de la nube es la manera de acelerar como desarrollamos nuevas aplicaciones, optimizar las existentes e interconectarse entre sí.

La tecnología estrella que permitió el surgimiento del desarrollo de aplicaciones nativas en la nube son los contenedores, que despliegan una aplicación en un entorno

de ejecución totalmente autónomo. Los contenedores, por sí solos, no son parte suficiente , pero sí necesaria de un entornos productivos. [3]

Para entornos productivos, surgieron soluciones llamadas orquestadores de contenedores. Actualmente el más usado es Kubernetes, de acá en adelante lo llamaremos K8S.

La CNCF, NSA, la industria del software y la comunidad están haciendo grandes aportaciones para el crecimiento y mejora de k8s, como así a otros proyectos nativos de la nube. Estos nuevos proyectos permiten mejorar y potenciar un cluster de Kubernetes y su arquitectura. Por lo que podemos encontrar nuevas características, que nunca fueron adoptadas, ni pensadas para otros orquestadores. Por defecto, en K8s, contamos con una configuración general, que no necesariamente cumple con los requisitos básicos de seguridad, así como tampoco con los exigidos por certificaciones internacionales.

El proceso de asegurar un cluster de k8s (Hardening) consiste en tomar las buenas prácticas o estrategias sobre la configuración del clúster. Ya que la configuración por defecto nunca es la más apropiada para infraestructura crítica y productiva.

Cuando construimos algo, por ejemplo una casa o una máquina, solemos confiar en la experiencia previa obtenida en un trabajo similar. Gracias a la experiencia personal, grupal o basados en experiencias compartidas por terceros, sabremos qué soluciones funcionan y cómo podremos evitar problemas/fallas. En otras palabras usamos tanto soluciones existentes, como así también técnicas que han demostrado ser eficientes en otras áreas o contextos. [4]

Es importante destacar que la generación de conocimiento y compartir el mismo en comunidades permiten desarrollar cada vez mejores productos o servicios.

El presente trabajo de fin de máster, se propone investigar y desarrollar una solución de infraestructura como código para el despliegue de entornos nativos de la nube en clústeres de tipo EKS / AWS que cumplan con los de seguridad propuestos por organizaciones directrices en la temática como CNCF, NSA, entre otros.

1.2. **Estado del Arte**

La publicación del proyecto de código abierto Docker en 2013, trajo nuevamente a escena a una tecnología, muy poco utilizada de manera masiva hasta ese momento. Los contenedores formaban parte de soluciones privadas de software, pero no tenían un uso masivo por parte de la industria del software.

Adicionalmente Docker incluyó ciertas características muy convenientes para el desarrollo de software, como una API de administración y un repositorio de imágenes,entre otras tantas ventajas.

La publicación de una API abrió las puertas a una mayor automatización. Por otro lado, la posibilidad de compartir imágenes de manera masiva a través de Internet permitió a los desarrolladores adoptar rápidamente la tecnología.

Con el paso del tiempo, la comunidad de desarrolladores fueron adoptando Docker y sus bondades para empaquetar, distribuir y desplegar software. La opción de poder dotar a las aplicaciones de todas sus dependencias de software en una misma imagen o paquete, permitió facilitar la tarea de despliegue y transición de versiones ,tanto en entornos de producción como de pruebas. [5]

Docker creció vertiginosamente, y fué inevitable que algunos usuarios se sintieran frustrados, ya que dedican recursos al adoptar soluciones que al poco tiempo eran reemplazadas por otras. Por éste motivo se creó la Open Container Initiative (OCI), responsable de estandarizar los componentes fundamentales de los contenedores. [6]

Docker y otros proyectos aportaron nuevas características que permitían orquestar el despliegue, escalado y administración de contenedores. Requisitos necesarios para una correcta administración de trabajos en producción. Para solventar estos problemas surgieron los primeros orquestadores de contenedores. [7]

El desarrollo con microservicios, como arquitectura de software, comenzó a ser viable, gracias a los contenedores y los orquestadores.

Debido a que uno de los grandes desafíos, eran cómo coordinar y dar soporte a tantos nuevos componentes que en conjunto forman una aplicación o aplicaciones .[8]

Una de las herramientas más utilizadas, al día de hoy, es Kubernetes (k8s). Proyecto de código abierto liderado por Google. Kubernetes fue adoptado por la CNCF siendo su proyecto estrella, y software base para muchos otros proyectos que controla.

Al igual que Linux, que es de código abierto, K8s se puede distribuir de una infinidad de maneras. Los proveedores públicos de la nube, han desarrollado y mantienen su propia distribución de kubernetes. En la actualidad una de las distribuciones / servicio en la nube más utilizado es EKS de Amazon web services.

1.3. Objetivos del Trabajo

Objetivo General

- Desarrollar un framework, de código abierto, permitiendo el despliegue de entornos nativos de la nube en clústeres k8s (EKS), mediante el uso de infraestructura como código, facilita la incorporación de buenas prácticas de ciberseguridad y auditoría en la configuración de los entornos.

Objetivos específicos

1. Explorar las opciones de herramientas disponibles, en el mercado, de orquestación de contenedores con k8s.
2. Analizar buenas prácticas o recomendaciones sugeridas por la OWASP, NSA, AWS y la CNCF.
3. Estudiar el funcionamiento del servicio de nube EKS incluido en AWS.
4. Catalogar recomendaciones o buenas prácticas de seguridad que potencialmente se incluyan en la solución propuesta.
5. Definir las herramientas de automatización que servirán de soporte para la implementación.
6. Proveer, al administrador y desarrolladores del entorno, de herramientas para auditar y mantener la seguridad en todas las etapas del ciclo de vida del desarrollo del software.
7. Desarrollar el código de la infraestructura (codificación).
8. Despliegue de aplicaciones del negocio de manera más segura.
9. Automatizar el despliegue de herramientas de seguridad código abierto asegura los clústeres.
10. Ejemplificar el uso de la solución desarrollada.
11. Publicar el proyecto a la comunidad para que pueda ser utilizado y mejorado.

1.4. Limitaciones

- La solución sólo abarca una sola distribución de k8s (EKS).
- No se espera abarcar todas las apps de seguridad existentes, sólo un grupo de ellas.
- Gran parte del Control Plane de EKS es manejado por AWS, por lo que no tenemos injerencia en el presente trabajo.
- Se plantean buenas prácticas y aplicaciones de seguridad que son necesarias para que un entorno nativo de la nube pueda ser más seguro, pero en ningún

caso son suficientes y perdurables en el tiempo,sin la revisión e interpretación de los resultados informados por la aplicaciones , día a día, por parte de los administradores. La seguridad se construye día a día.

- El estudio de buenas prácticas y recomendaciones se limita a OWASP, NSA, AWS y la CNCF.

1.5. Impacto en sostenibilidad, ético-social y de diversidad

En 2015, la Organización de las Naciones Unidas aprobó la agenda 2030 sobre el desarrollo sostenible. La agenda consta de 17 objetivos de desarrollo sostenible (ODS), tanto España, como la UOC proponen a sus miembros proponer y evaluar acciones que colaboren con la concreción de los ODS.

Las aplicaciones nativas en la nube, como su infraestructura no se encuentran exentas de la realidad mundial.

La naturaleza del proyecto , de código abierto, pondrá al alcance de la comunidad una solución qué puede ser utilizada como guía para personas y organizaciones. Fomenta la reducción de la desigualdades en el acceso a herramientas de esta naturaleza (ODS 10).

El noveno objetivo número 09 de Industria, innovación e infraestructura, también se ve beneficiado por el desarrollo tecnológico en tecnologías de virtualización y computación, que permiten potenciar las industrias y reducir los costos de infraestructura física , como así también el uso de recursos naturales cómo el agua o la energía (ODS 11). La reducción de desechos tecnológicos derivados de la actividad aporta a cumplir el ODS 11 de ciudades y comunidades sostenibles. Un uso más moderado de los recursos naturales y una menor generación de desechos también aporta al objetivo número 13 (ODS 13) sobre el cambio climático.

1.6. Enfoque y método seguido

El presente trabajo forma parte del ámbito empresarial, el cuál consiste en investigar las mejores prácticas, recomendaciones y estrategias para asegurar, auditar y desplegar clústeres de k8s

La falta de conocimiento de los detalles y características del producto final, nos sugiere utilizar una metodología de tipo ágil o iterativa incremental. Ya que el proyecto es desarrollado por una sola persona, prescindiremos de elementos ceremoniales, pero conservamos la planificación por tareas / casos de usos resueltos en sprints de tamaño fijo.

Podemos identificar etapas globales del proceso como lo son las siguientes:

- Descubrimiento + Investigación
- Diseño
- Desarrollo
- Memoria final y defensa

En los sprints iniciales será de descubrimiento de la temática, para un posterior diseño de la solución propuesta. En los siguientes, se desarrolla y codifica. Una vez completa, se documenta una prueba de concepto.

Completada las etapas indicadas, se procederá a documentar los resultados en una memoria del trabajo y todos los requerimientos para la defensa del trabajo final.

De acuerdo a la naturaleza del trabajo final y sus puntos de revisión, se establecerán 3 Sprint de investigación y desarrollo, coincidiendo con los PEC 1 ,2 y 3. El cuarto corresponde al PEC 4 para la redacción de la memoria.

1.7. Planificación del Trabajo

1.7.1 Administración de proyecto

Como herramienta soporte para la administración del proyecto se selecciona Trello.[9] La misma permite generar tareas , categorizarlas, almacenar la información relacionada al respecto en la misma. También ofrece la posibilidad de generar un calendario de ejecución, cómo así también diagrama de Gantt .

Es importante destacar que Trello, permite visualizar las tareas de manera temporal, como así también por su estado. En el presente trabajo se utilizarán 4 estados posibles.

- **Backlog del producto:** Lista ordenada de todo los requerimientos para completar el producto. [10] Incluye tareas, características o defectos a ser desarrollados.
- **En progreso:** Lista de tareas que se encuentran en ejecución actualmente.
- **Completada:** Lista de tareas completadas.
- **Bloqueada:** Tareas ,que por algún motivo, dependen de la realización de otra tarea para poder comenzar su desarrollo.

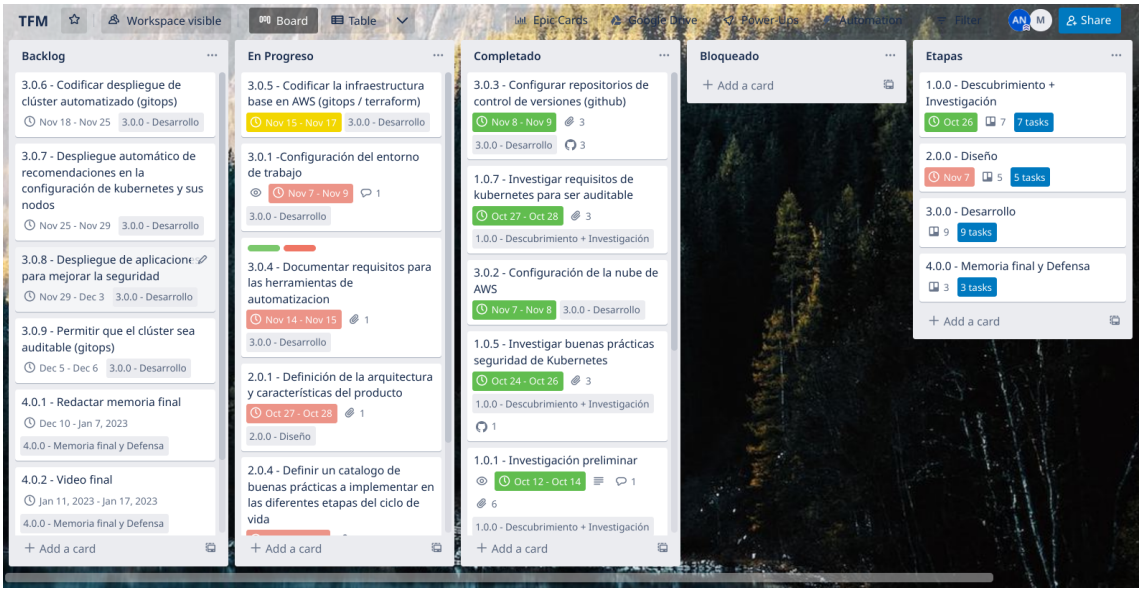


Imagen Nº 1: Despliegue del tablero trello

1.7.2 Recursos

- Ordenador
- Internet
- Acceso Amazon web services
- Cuenta de github para el código
- Cuenta de Trello

1.7.3 Listado de Tareas

ID	Tareas	Dependencias	Fecha inicio	Fecha final	Dias	Completado
1.0.0	**Descubrimiento + Investigación**		15/03/2023	31/03/2023	14	100,00 %
1.0.1	Investigación preliminar		15/03/2023	17/03/2023	2	100,00 %
1.0.2	Investigar Distribuciones de Kubernetes		16/03/2023	20/03/2023	1	100,00 %
1.0.3	Investigar Amazon Web Services		19/03/2023	20/03/2023	1	100,00 %
1.0.4	Investigar herramientas para la automatización y GitOps		20/03/2023	22/03/2023	2	100,00 %
1.0.5	Investigar buenas prácticas seguridad de Kubernetes		23/03/2023	26/03/2023	2	100,00 %
1.0.6	Analizar el ciclo de vida aplicaciones nativas del cloud		24/03/2023	25/03/2023	1	100,00 %
1.0.7	Investigar requisitos de kubernetes para ser auditable		22/03/2023	26/03/2023	2	100,00 %
2.0.0	**Diseño**	**1.0.0-1.0.7**	01/4/2023	10/04/2022	11	100,00 %
2.0.1	Definición de la arquitectura y características del producto		03/04/2023	05/04/2023	1	100,00 %
2.0.2	Seleccionar lenguajes y herramientas de automatización		04/04/2023	05/04/2023	1	100,00 %
2.0.3	Gestión de la configuración del proyecto	2.0.2	06/04/2023	06/04/2023	1	100,00 %
2.0.4	Definir un catálogo de medidas a implementar en las diferentes etapas del ciclo de vida		07/04/2023	07/04/2023	1	100,00 %
2.0.5	Definir un catálogo de aplicaciones de seguridad externas a incluir		10/04/2023	10/04/2023	3	100,00 %
3.0.0	**Desarrollo**		11/04/2023	09/05/2023	0	90%

3.0.1	Configuración del entorno de trabajo		11/04/2023	12/04/2023	2	100%
3.0.2	Configuración de la nube de AWS		13/04/2023	13/04/2023	1	100%
3.0.3	Configurar repositorios de control de versiones (github)	2.0.3	14/04/2023	14/04/2023	1	100%
3.0.4	Documentar requisitos para las herramientas de automatización	2.0.2	15/04/2023	15/04/2023	1	100%
3.0.5	Codificar la infraestructura base en AWS (GitOps / terraform)	3.0.4	15/04/2023	25/04/2023	7	100%
3.0.6	Codificar despliegue de clúster automatizado (GitOps)	3.0.5	25/04/2023	28/04/2023	3	90%
3.0.7	Despliegue automático de recomendaciones en la configuración de kubernetes y sus nodos	3.0.6	01/05/2023	05/05/2023	7	100%
3.0.8	Despliegue de aplicaciones para mejorar la seguridad	3.0.6	08/05/2023	10/05/2023	3	100%
3.0.9	Permitir que el clúster sea auditable (GitOps)		10/05/2023	11/05/2023	1	100%
3.0.10	Pruebas y correcciones		12/05/2023	22/05/2023	4	
4.0.0	**Memoria final y Defensa**		10/05/2023	30/06/2023	47	
4.0.1	Redactar memoria final		10/05/2023	13/06/2023	35	
4.0.2	Video final		14/06/23	20/06/2023	7	
4.0.3	Defensa TFM		27/06/2023	30/06/2023	5	

Tabla Nº 1 : Listado de tareas y su calendarización

1.7.4 Calendario (Diagrama de Gantt)

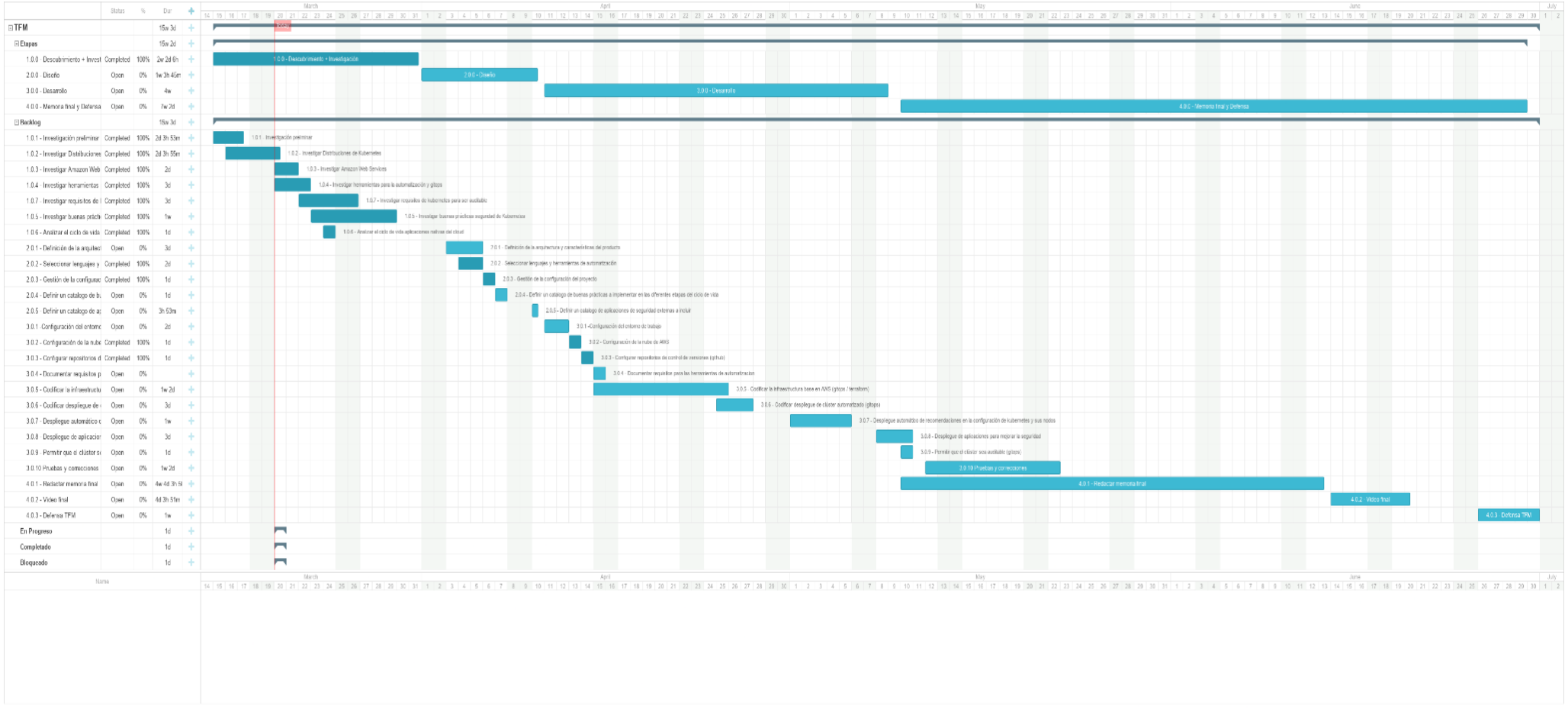


Imagen Nº 2: Diagrama de Gantt

1.8. Breve sumario de productos obtenidos

El presente trabajo final de máster está compuesto por los siguientes productos.

- Informe final / memoria con un detalle teórico de los conceptos teóricos básicos requeridos, como así también detalles.
- Descripción de las características desarrolladas e incluidas en el framework.
- Repositorios de github públicos.
- Ejemplificación del uso.

1.9. Breve descripción de los otros capítulos de la memoria

La memoria del trabajo final de máster consta de 9 capítulos de los cuales podemos destacar los siguientes:

- Capítulo 2: Desarrollo teórico de conceptos básicos en la temática.
- Capítulo 3: Se realiza un desarrollo teórico que explica las tecnologías más importantes para la construcción del framework.
- Capítulo 4: Desarrollo teórico de componentes y tecnologías relacionadas con la seguridad, a diferencia del anterior, es exclusivo para seguridad.
- Capítulo 5: Explicación detallada de todas las características que incluye el framework desarrollado, y sus posibles aplicaciones.
- Capítulo 6: Ejemplificación de cómo nos puede ayudar el framework en ciertos escenarios.

El resto de capítulos no mencionados, en la lista anterior, fueron excluidos intencionalmente, ya que son comunes a cualquier trabajo de investigación, incluidos en la plantilla de memoria provista por la universidad. Por lo tanto no requieren un desarrollo particular

2. Conceptos básicos

Para la realización del presente trabajo final, fue necesario investigar , aprender y relacionar diferentes tecnologías y conceptos, que forman parte de la solución diseñada y propuesta.

2.1 Arquitectura de microservicios

Sam Newman en su libro “Building Microservices” 2da edición afirma que las arquitecturas de microservicios han incrementado su popularidad en los últimos años, su naturaleza la convierte en una excelente forma de diseñar software y poder tomar las ventajas que provee las nubes computacionales [11]

Los microservicios son servicios independientes, los cuales son modelados de acuerdo al dominio del negocio. Los servicios encapsulan funcionalidad y la ponen a disposición de los otros servicios por medio de la red de ordenadores. El conjunto de las funciones encapsuladas en los microservicios constituyen una aplicación o sistema mayor.

Los microservicios son como cajas negras, que ocultan su funcionalidad, y solo proveen una sola vía de comunicación por medio de una API REST. También los microservicios suele utilizar bases de datos exclusivas y no compartidas

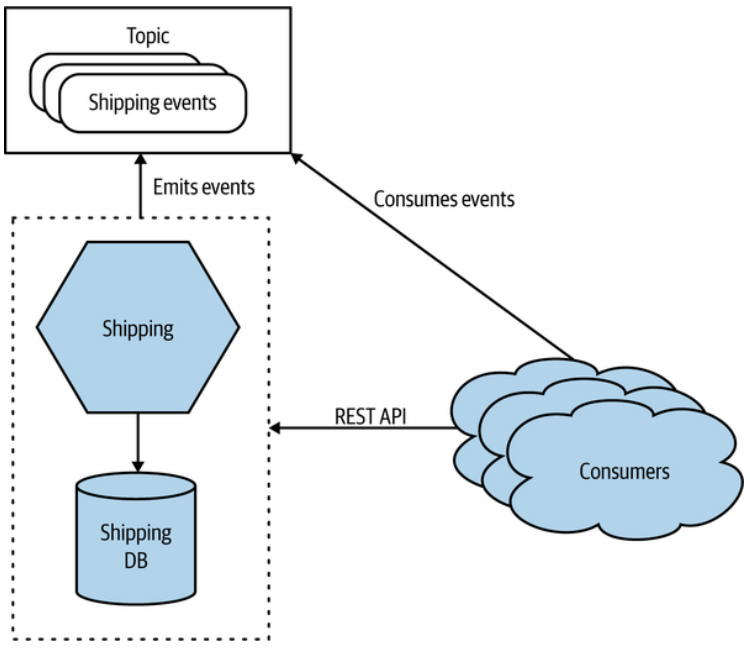


Imagen Nº 3: Microservicio “Shipping/Envios” exponiendo su funcionalidad

Las aplicaciones con arquitecturas monolíticas lentamente están siendo reemplazadas por las basadas en microservicios.

Despliegue independiente: En contraposición de una arquitectura monolítica, un microservicio puede ser actualizado (despliegue de una versión diferente) sin

necesidad afectar otros servicios activos. En entornos orquestados, incluso es posible transicionar entre versiones sin afectar el servicio, gracias a las estrategias actuales de despliegue.

Maneja su propio estado: Cada servicio debe almacenar, ocultar y manejar su propia información, si otro servicios necesita información dentro del dominio de otro servicio, no se debe consultar directamente a la base de datos, sino la consulta debe ser realizada por medio de la interfaz publicada (API). En pocas palabras cada microservicio es responsable de publicar o no la información y también determinar quienes pueden acceder a ella .

Tamaño: No existe una regla general para determinar el tamaño correcto de un microservicio, la percepción de tamaño es muy subjetiva. Sam Newman en su libro Building microservices afirma que un microservicio debe tener un tamaño que sea fácil de comprender. Sin embargo, es también concentrarnos en cuanto servicios podemos administrar actualmente. En regla general cada microservicio agrega una complejidad adicional a la hora de administrar, monitorear y desplegar los servicios.

2.1.2 Microservicios en contenedores y kubernetes

Existen múltiples opciones para el despliegue de microservicios, comenzando en máquinas físicas sin automatización, pasando por virtualización y contenedores.

Los contenedores son la mejor solución para el despliegue de microservicios, Docker nos permite garantizar una ejecución aislada, además permite automatizar mediante el uso de diferentes tecnologías . Kubernetes sumado a Docker nos permite cumplir con principio como 0 caídas de servicio en las actualizaciones (estrategias de despliegue) , escalar , y monitorear el estado de los servicios.

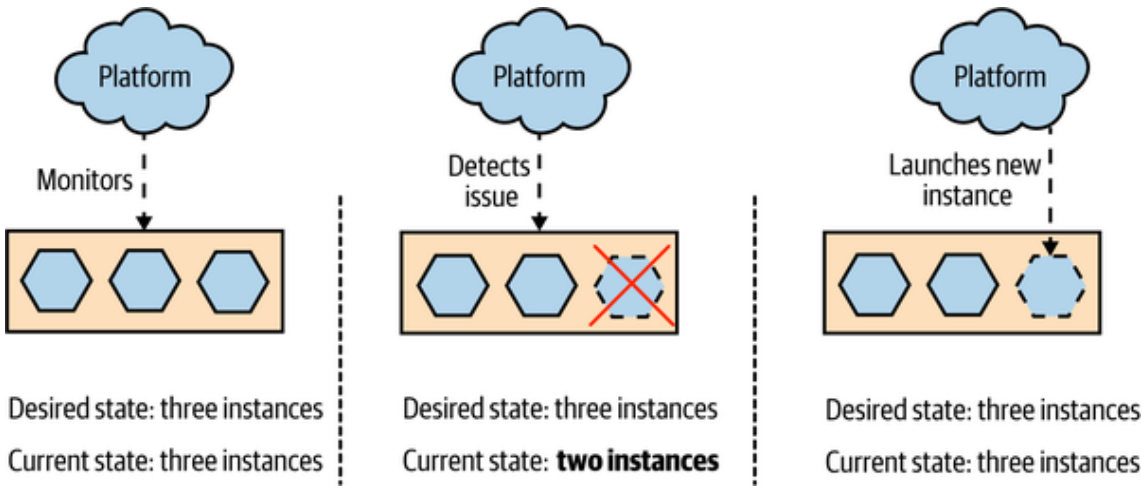


Imagen Nº 4: Funcionamiento orquestador en caso de fallos en una réplica.

K8s o cualquier otro orquestador de contenedores debe ser capaz de ejecutar servicios con múltiples instancias por detrás. Las instancias deben ser constantemente monitoreadas, para poder actuar cuando alguna instancia no funcione de forma correcta. La acción por defecto es matar el contenedor y levantar uno nuevo.

2.2 Infraestructura como código / Integración continua

2.2.1 Infraestructura

Rosemary Wang en su libro Infrastructure as Code, Patterns and vPractices define a la infraestructura como software, plataformas o hardware que entrega y/o despliegan aplicaciones en producción.[12] Tradicionalmente la infraestructura sólo estaba formada por recursos físicos computacionales, de redes y almacenamiento, donde se ejecutaban las aplicaciones. Con la aparición de la nube y nuevas plataformas el concepto de infraestructura ha cambiado radicalmente.

Ejemplo de elementos de infraestructura

- Servidores
- Plataformas de orquestación (kubernetes)
- Balanceadores de carga
- Conmutadores de red
- Base de datos
- Colas
- Cache

Infraestructura como código, actualmente aplica prácticas DevOps para automatizar el proceso de cambios en la infraestructura de manera de lograr escalabilidad, resiliencia y seguridad.

2.2.2 Configuración manual

Cada componente de una infraestructura dispone de una configuración por defecto, que en la mayoría de los casos no satisface los requerimientos mínimos iniciales. La introducción de cambios manuales (cambios ad hoc), en componentes de infraestructura, históricamente introdujo grandes desafíos para mantener y escalar los sistemas y los equipos de trabajo. La ejecución de cambios ad hoc de configuración incrementan la tasa de fallos, y no permite disponer de un código base que permita reproducir los cambios de manera automática, rápida y con una menor probabilidad de introducir errores humanos.

Replicar infraestructuras de manera manual y su correcta configuración es una tarea compleja y propensa a errores.

2.2.3 Desvíos en la configuración (Configuration Drift)

Desviación / cambios de configuración del estado deseado de la infraestructura y la configuración actual. La IaC considera el código de configuración almacenado en el repositorio de código como la fuente de la verdad. Sin IaC y sus herramientas resulta una tarea compleja, asegurar la ausencia de desvíos de configuración.

2.2.4 Código fuente (repositorio de código) la fuente de la verdad

Un repositorio de tipo Git es una base de datos clave valor, que contiene toda la información necesaria para mantener y administrar las revisiones y la historia de los ficheros del proyecto. [13]

Actualmente los beneficios de los repositorios de código son ampliamente valorados en la ingeniería del software, ya que la naturaleza de los proyectos de software exigen la posibilidad de incorporar tecnologías o herramientas que apoyen el trabajo en equipo.

Los repositorios de código, permiten almacenar, revisar, versionar archivos de textos. además agregan la posibilidad de leer de una manera simple todo el historial de cambios del fichero. Por todo lo expuesto anteriormente, los repositorios, son el lugar ideal para guardar la configuración y considerarlo la fuente de la verdad.

2.2.5 Ciclo de vida del desarrollo IaC

Las herramientas de IaC permiten describir la configuración y despliegue de los diferentes componentes de la misma, en un lenguaje de programación declarativo. Al igual que las aplicaciones, IaC requiere un ciclo de vida de desarrollo.

El flujo tradicional de IaC requiere el desarrollo de uno o varios ficheros de configuración o scripts de infraestructura. Los ficheros generados se almacenan en repositorios de control de versiones, Ej Git. Tan pronto como el commit se encuentre en el repositorio, se ejecutarán controles automatizados.

En la IaC necesitamos de herramientas que puedan leer los ficheros de configuración, interpretarlos y desplegarlos. Para este fin existen herramientas de código abierto que nos facilitan la tarea.

Como último paso es importante probar si los recursos desplegados fueron correctamente creados.

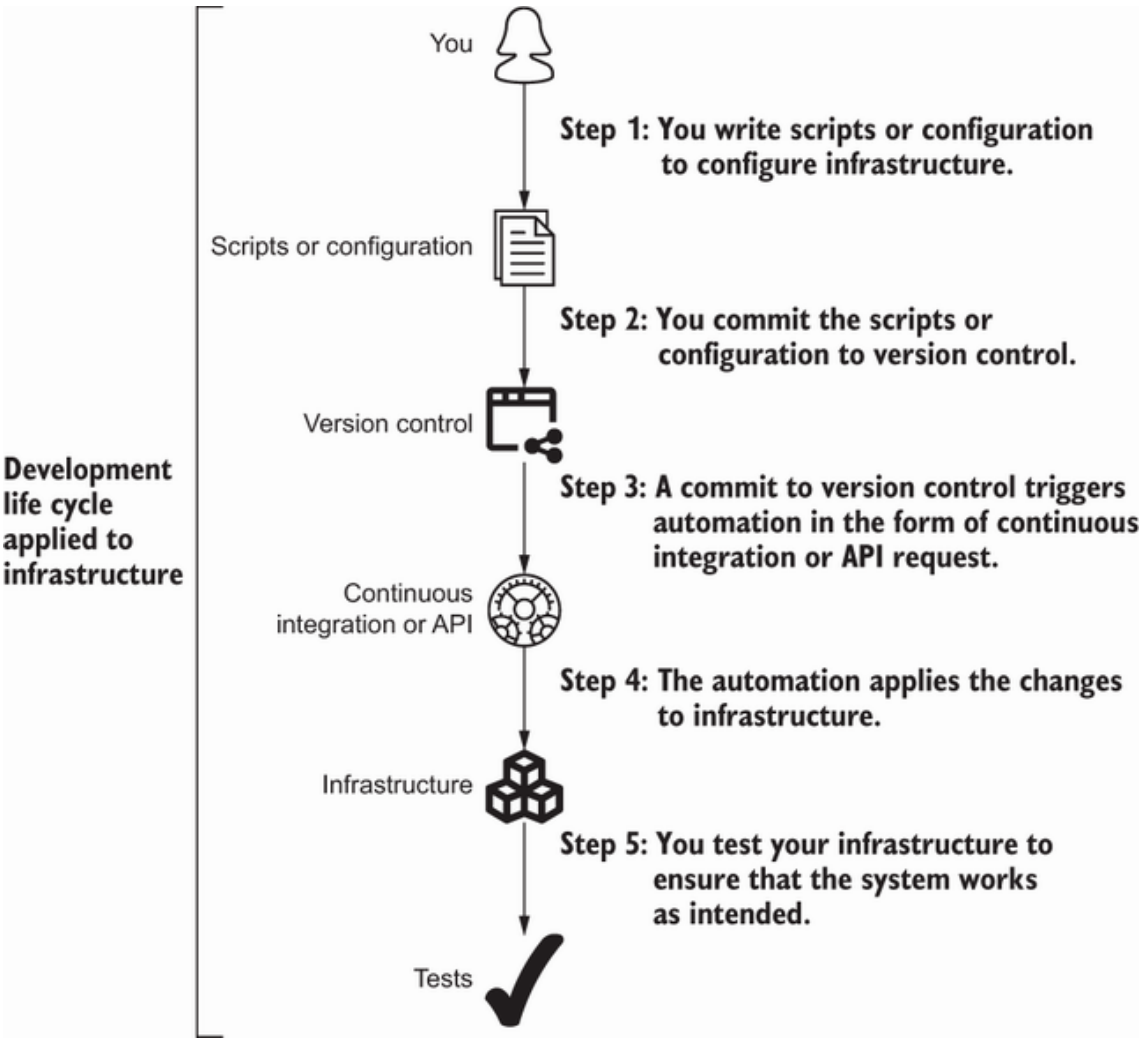


Imagen Nº 5: Ciclo de vida del desarrollo de infraestructuras

2.2.6 Principios de IaC

No cualquier código o configuración relacionados a infraestructura puede garantizar adecuada escalabilidad o evitar caida de servicios

- **Reproducibilidad:** El codigo de configuracion o descripción de infraestructura puede ser utilizado para generar un nuevos entornos o recursos de infraestructura
- **Idempotencia:** Se debe garantizar que la ejecución de cada automatización produzca siempre el mismo resultado. Para garantizar el despliegue del estado deseado, es muy importante, asegurar que nuestro código siempre produce la misma configuración
- **Componibilidad:** Ensamblar múltiples recursos de infraestructura y su actualización no debe afectar los demás componentes. La modularidad nos permite lograr la independencia entre grupos de componentes sin acoplamiento
- **Evolucionabilidad:** El desarrollo de infraestructuras debe contemplar y facilitar cambios en los recursos de infraestructura, minimizando el esfuerzo y riesgo de fallo.

2.3 Nube

2.3.1 ¿Qué es la nube?

La computación en la nube pública son plataformas que ofrecen recursos computacionales por internet. En lugar de comprar y mantener equipos físicos en los centros de datos, podemos acceder a servicios computacionales, almacenamiento y base de datos por un precio a demanda, facturados por una unidad de medida de uso. Normalmente la métrica más utilizada es el tiempo de vida del recurso, pero debido a los diversos tipos de recurso de naturaleza radicalmente diferentes, otras métricas pueden ser consideradas. Las plataformas más conocidas son Amazon web services, Google cloud , Microsoft Azure, Ali cloud.

Organizaciones de todo tipo , tamaño e industria utilizan la nube en una gran cantidad de casos de usos, tales como copias de seguridad, recuperación en desastre , correo electrónico, escritorios virtuales, desarrollo y pruebas de software. Por ejemplo, compañías médicas utilizan la nube para desarrollar tratamientos personalizados para pacientes. [14]

2.3.2 Características de la nube

Los servicios de la nube, actualmente, deben disponer de las siguientes características como mínimo.[15]

Bajo demanda y autoservicio: Se debe permitir al usuario final la capacidad de aprovisionar recursos computacionales por sí mismo, sin necesidad de afrontar los retrasos inherentes del modelo on premise, donde debíamos hacer pedidos de software y hardware, era muy importante considerar los tiempos de aprovisionamiento para la planificación de proyecto de software.

Hardware compartido: Los proveedores de nube basan sus servicios en recursos compartidos, un componente de hardware es compartido por varios clientes. El proveedor de la nube debe garantizar un aislamiento lógico entre clientes. Sin embargo, también ofrecen equipamiento dedicado para casos excepcionales que no permitan su ejecución en hardware compartidos .

Elasticidad: La elasticidad es una característica fundamental en los proveedores de nube. Se encuentra muy relacionada con la capacidad de obtener recursos extras bajo demanda. Las aplicaciones pueden crecer o decrecer su capacidad computacional requerida, de acuerdo a las necesidades actuales de demanda de los usuarios. En modelos on premise, deberíamos calcular el máximo de recursos que podrían ser necesarios, comprar el hardware y software, y desplegarlo. La contra del modelo on premise es que debemos mantener una capacidad ociosa y

pagar por ella aunque no la hemos ciertamente utilizado. Otra característica importante es que el usuario pueda pagar solo por el tiempo utilizado de los recursos o equivalente en otra métrica de facturación.

2.3.3 Modelo de responsabilidad compartida

El 45 % de las filtraciones de datos ocurrieron en la nube, según un estudio de IBM security ¨Costo de las filtraciones de datos 2022",[16] los proveedores de nube públicos son una excelente opción en la cuál confiar la seguridad de nuestra infraestructura.

Una gran ventaja de los proveedores de nube, es que no necesitamos preocuparnos por la seguridad de nuestra infraestructura subyacente, en su lugar, podemos dedicar más tiempo para desarrollar y asegurar nuestros flujos de trabajo con un nivel más alto de abstracción. Sin embargo, debemos conocer a detalle el modelo de responsabilidad compartida propuesto por el proveedor, ya que los proveedores de nube, no pueden asegurar al 100% nuestros flujos de trabajo.[17]

Por ejemplo en AWS, los usuarios tienen la responsabilidad de configurar apropiadamente el control de acceso mediante Identity and Access Management (IAM), EC2 security groups, etc. En otras palabras, la seguridad del software y hardware de la plataforma es responsabilidad de AWS, pero la configuración de usuario final y la del software implementado es exclusiva responsabilidad del usuario.

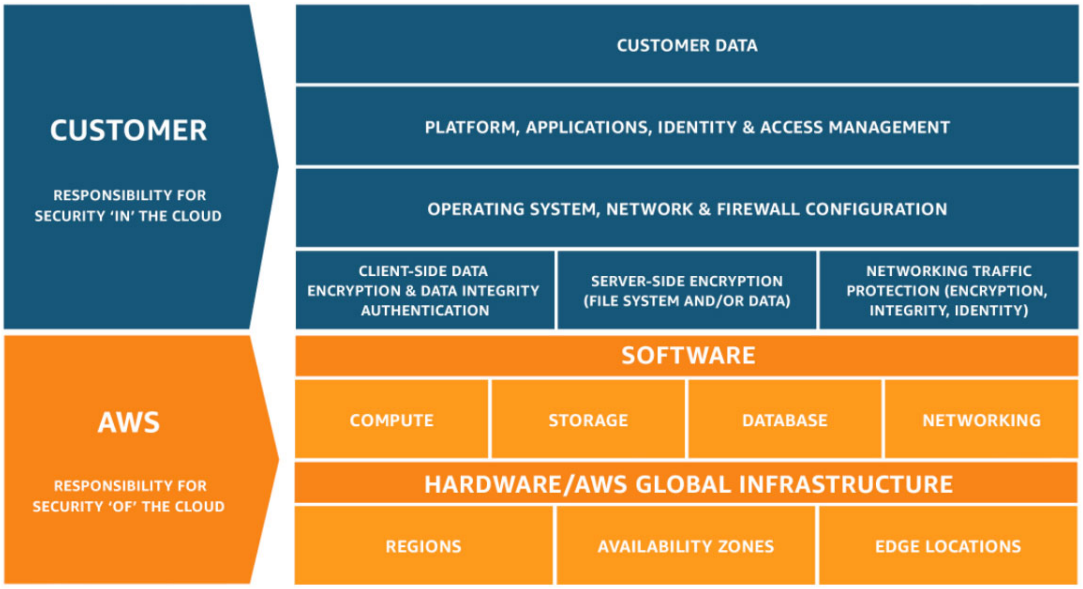


Imagen Nº 6: Distribución de responsabilidades entre el cliente y AWS

Los proveedores de nube cumplen con los requerimientos exigidos por las normas de certificación internacional en diferentes industrias. Ej PCI (Pago con tarjeta de crédito)

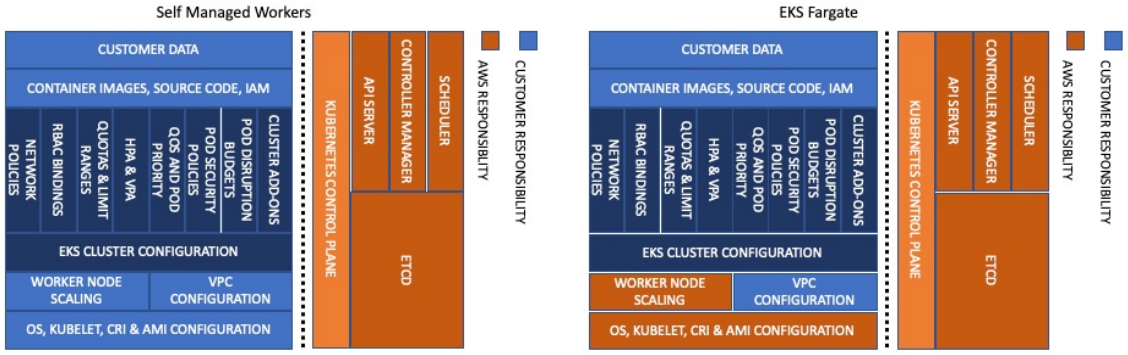


Imagen Nº 7: Distribución de responsabilidad para el servicio de k8s en AWS

La división de responsabilidades en Amazon EKS puede considerarse de dos formas. Si utilizamos AWS fargate , el usuario es abstraído del mantenimiento de los nodos del cluster. Si no utilizamos AWS fargate, la responsabilidad de mantener actualizado los nodos (parches de seguridad) recae en el usuario. [18]

En la imagen anterior, se puede distinguir la asignación de responsabilidades de acuerdo a los diferentes tipos de despliegue del cluster.

La responsabilidad compartida es evidente e inevitable.

2.4 Nativos en la nube

Las nuevas organizaciones que han comenzado a construir tecnologías con DevOps y Cloud, y son consideradas como organizaciones nativas del cloud. Los desarrolladores incluyen entre sus tareas , monitoreo de tableros de control , guardias activas, para facilitar resolución de incidentes cuando los servicios fallan. Los equipos de operaciones concentran sus acciones en construir la plataforma en la que opera el software.

Organizaciones preexistentes a la era de la nube, adaptan ciertos equipos para afrontar la transformación digital. [19]

En la siguiente gráfica, tomada del libro Cloud Native Application Security [Guy Podjarny,2021] muestra como la distribución de responsabilidades cambia con la adopción del cloud.

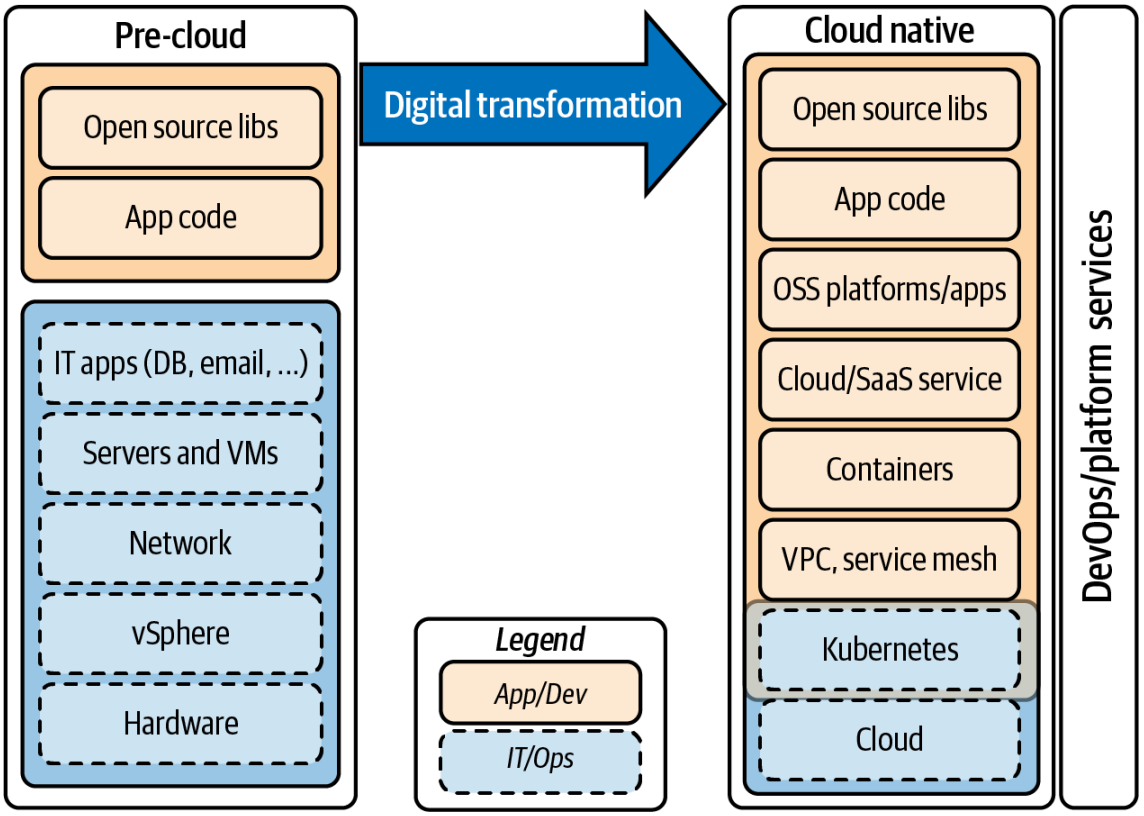


Imagen Nº 8: Transformación digital de las organizaciones

Los equipos de desarrolladores han tomado más protagonismo en tareas que antes eran ajenas, por lo que las organizaciones pre nativas al cloud han debido diseñar sus equipos de nuevo (IT/Ops/App/dev).

2.4.1 Aplicaciones nativas de la nube

CNCF es una organización creada por la Linux Foundation cuya misión es crear e impulsar la adopción de un nuevo paradigma en computación.[20]

La CNCF actualmente administra una docena de proyectos de código abierto tales como K8s , coredns, prometheus , etc.

Las tecnologías nativas del cloud facultan a las organizaciones a crear y ejecutar aplicaciones escalables en entornos modernos y dinámicos como los proveedores de nube.

Contenedores , servicios mesh , microservicios , infraestructuras inmutables , y API declarativas ejemplifican este enfoque. En el presente trabajo solo revisaremos algunos casos

CNCF ayuda a la adopción de técnicas que buscan un bajo acoplamiento en los sistemas, lo cual permite aplicaciones más resilientes, administrables, monitoreables, automatización robustas. Todo esto permite a los ingenieros y sus organizaciones hacer cambios significativos de gran impacto de manera más frecuente y predictiva.

Guy Podjarny, en su libro Cloud Native Application Security (2021) afirma que en las aplicaciones nativas, Hardware y las máquinas virtuales son reemplazados por contenedores manejados a través de Dockerfiles almacenados en repositorios de

código fuente; La configuración de red es declarada como ficheros de infraestructura como código Ej. Terraform y Kubernetes; Base de datos son provistas como servicio por el proveedor de nube, etc. [21]

En el presente trabajo nos centraremos en el desarrollo de plataformas seguras (Kubernetes/AWS) para la ejecución de de aplicaciones nativas en la nube, por ello tomaremos un tiempo en revisar las bases conceptuales de la tecnología requeridas

3. Tecnologías

3.1 Contenedores y Virtualización

Los contenedores y las máquinas virtuales son dos tecnologías de virtualización diferentes, sin embargo para facilitar la comprensión de los conceptos se las suele comparar.

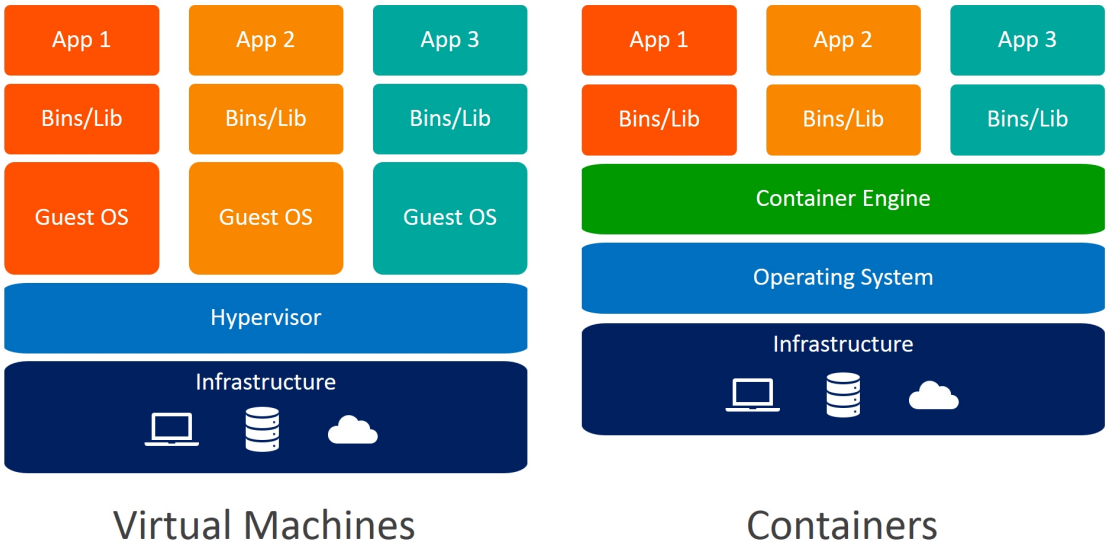


Imagen Nº 9: Esquema comparativo máquinas virtuales y contenedores

Cada máquina virtual instala un sistema operativo, sin embargo los contenedores interactúan directamente con el Kernel del sistema operativo.

Los contenedores y las máquinas virtuales son diferentes formas de desplegar aplicaciones dentro de entornos aislados del hardware subyacente. La clave es el nivel de aislamiento. [22]

Los contenedores son más eficientes en consumo de recursos computacionales, ya que comparten el núcleo del sistema operativo con la máquina anfitriona.

Las buenas prácticas de creación de imágenes enfatizan en crear imágenes que incluyan lo mínimo y necesario, concepto que no es posible de cumplir con las máquinas virtuales. Por otro lado, las máquinas virtuales proveen un mayor aislamiento de la máquina anfitriona y de sus vecinos. La máquina anfitriona no tiene control sobre lo que es ejecutado en las máquinas virtuales. Para remediar la falta de control o visibilidad se debe incluir en las máquinas virtuales agentes de aplicaciones de seguridad que reporten desviaciones de seguridad.

Los contenedores proveen características de seguridad que aíslan los contenedores de la máquina que lo aloja. Sin embargo, si un atacante se aprovecha de alguna vulnerabilidad del kernel, relativo al aislamiento, podría llegar a tener acceso

a la máquina anfitriona. Por ello se recomienda mantener actualizado el software y prestar especial atención a los reportes de vulnerabilidad de día cero.

3.2 Docker

Un contenedor consiste en un paquete que incluye código de aplicación, binarios, archivos de configuración y librerías, todo junto, en un simple paquete llamado imagen de contenedor. Rápidamente se puede empaquetar y probar contenedores creados localmente, ya que podemos conocer exactamente la composición del entorno de ejecución y siempre será el mismo tanto en local , como en cualquier otra máquina de la misma arquitectura de procesador.

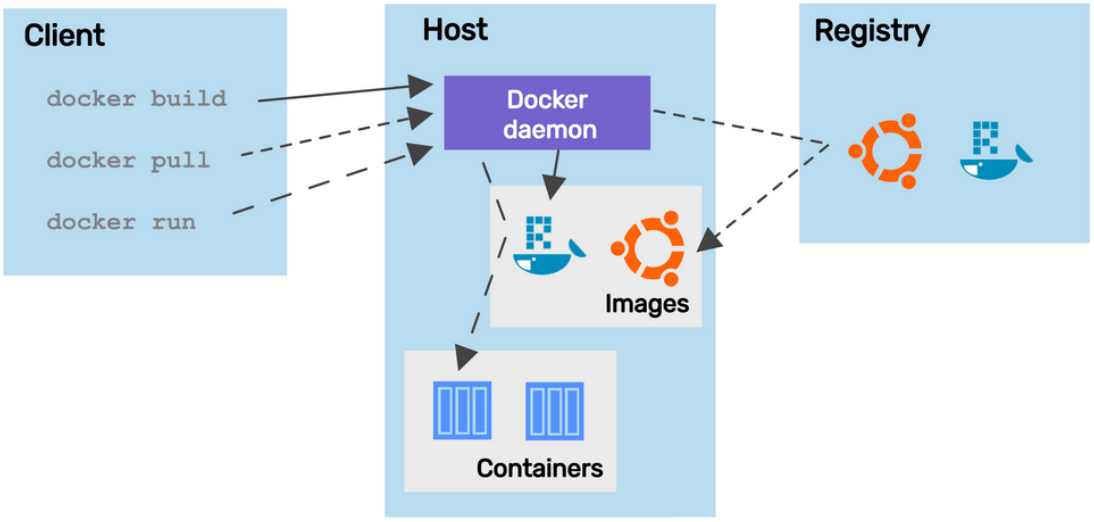


Imagen Nº 10: Arquitectura cliente servidor de Docker

Docker usualmente es considerado como un componente atómico que permite administrar, ejecutar y crear contenedores . Sin embargo la arquitectura de docker es modular. [23]

3.2.1 Componentes del motor de docker

- Demonio de docker
- containerd
- runc
- Plugin

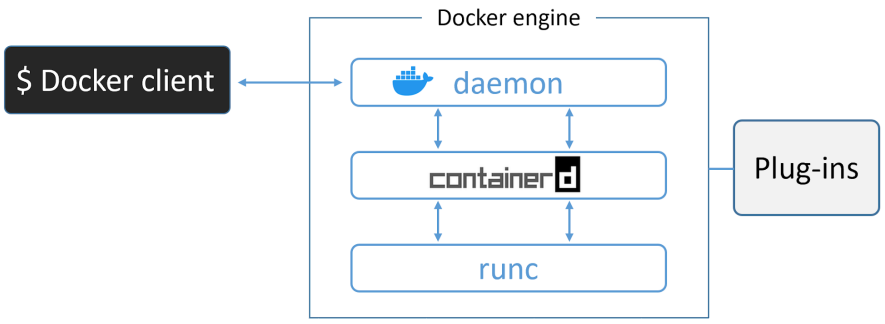


Imagen Nº 11: Vista de componentes de Docker

3.2.2 Demonio de Docker

El motor de docker es el software núcleo que ejecuta y administra los contenedores, en previas versiones el demonio la especificación era monolítica, su naturaleza generó desafíos a resolver tales como

1. Es complicado innovar por su alto acoplamiento, y la introducción de cambios afectaron más otros componentes
2. Más lento , ya que incluye nuevos componentes que no son necesarios en todos los casos
3. No aprovecha de las ventajas que obtienen las aplicaciones nativas del cloud

En las últimas versiones se propone el siguiente desacoplamiento

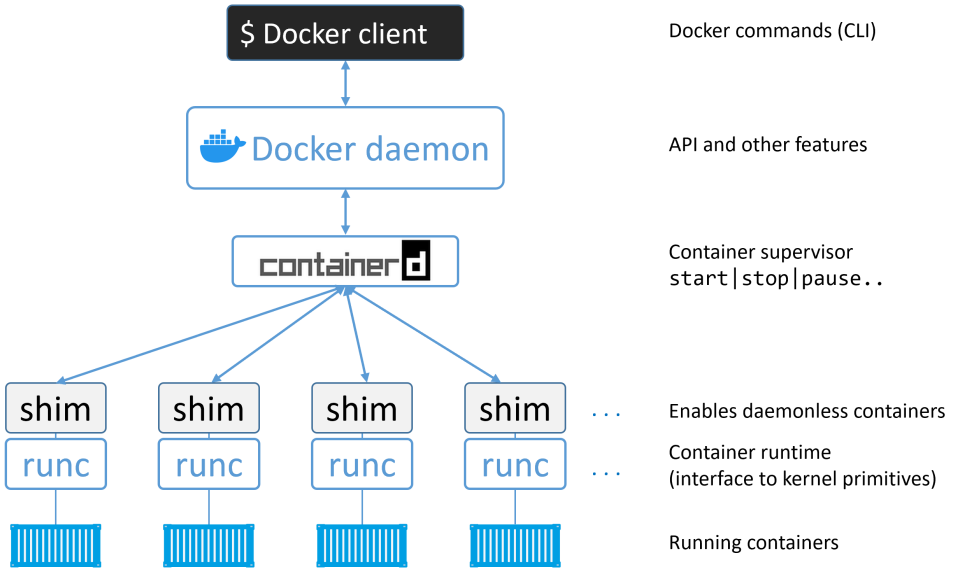


Imagen Nº 12: Esquema jerárquico de composición de docker

3.2.3 containerd & runc

Containerd y runc incluye toda la lógica de ejecución de contenedores, funciones como start , stop, pause y rm son parte de containerd. También se incluye manejo de imágenes , volúmenes y redes .

Containerd envía la solicitud de creación de un contenedor en formato OCI. Runc interactúa directamente con el sistema operativo para la creación del container.

3.2.4 Imágenes

Las imágenes de Docker es un empaquetado que contiene todo lo necesario para que una aplicación pueda ejecutarse.

Las imágenes pueden incluir

- Código fuente de las aplicaciones
- Dependencias, de software, de las aplicaciones
- componentes de sistemas operativos

Para facilitar la comprensión del concepto de imagen, podemos compararlo con una plantilla de una máquina virtual, la cual , nos permite crear VM exactamente iguales.

Las imágenes pueden ser alojadas en registros, desde donde se las puede obtener. Los registros pueden ser públicos o privados. El repositorio público oficial del proyecto Docker es **Docker Hub**, pero al día de hoy, existen muchas más opciones.

En los repositorios públicos podemos encontrar una gran cantidad de imágenes listas para usar publicadas por la comunidad.

3.2.5 Imágenes y sus capas

Técnicamente una imagen de docker está compuesta por múltiples capas apiladas de solo lectura, dentro de las capas podemos encontrar archivos relacionados al sistema operativo, código fuente, dependencias, etc.

Las capas son ubicadas en forma de pila, y cada nueva capa es agregada sobre la última capa. Como buena práctica las imágenes deben ser pequeñas y de pocas capas, para poder obtener contenedores rápidos y ligeros.

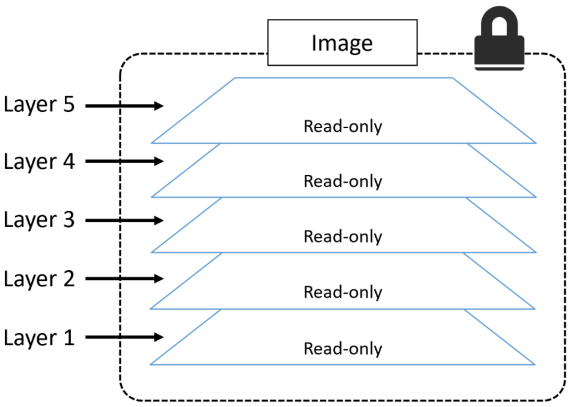


Imagen Nº 13: Ejemplo de imagen compuestas por diferentes capas

En la siguiente imagen, se visualiza como los archivos generados en la imagen fueron ubicados en las diferentes capas.

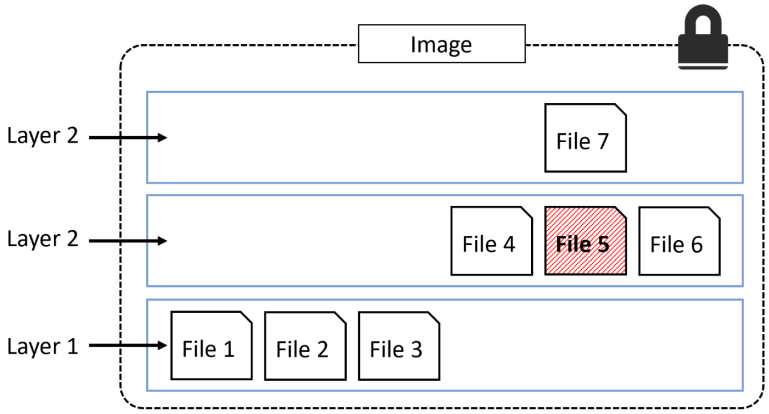


Imagen Nº 14: Muestra la distribución de archivos dentro de las capas

Docker utiliza diferentes drivers que leen las imágenes y unifican las diferentes capas en un solo objeto de tipo sistema de archivo. Ej: AUFS, overlay, etc.

3.2.6 Escribiendo imágenes

Las imágenes de contenedores son un componente fundamental en las aplicaciones nativas de la nube. Cada imagen debe contener los archivos y dependencias necesarias para poder ser ejecutada la aplicación. Docker provee una herramienta para interpretar los manifiestos que describen cada capa.

Los manifiesto son llamados Dockerfile, la primera línea del manifiesto debe referenciar una imagen existente que se usará como base. La imagen base suele ser

una imagen que incluye componentes específicos de un sistema operativo. Cada distribución del sistema operativo mantiene y publica regularmente las imágenes actualizadas del sistema operativo. [24]

A continuación ejemplo de un Dockerfile

```
FROM node:18-alpine
WORKDIR /app
COPY . .
RUN yarn install --production
CMD ["node", "src/index.js"]
EXPOSE 3000
```

FROM: Define la imagen base a utilizar.
WORKDIR: Define el directorio, en árbol virtual del sistema de archivo, donde se ubicará la aplicación.
COPY: Copia archivos en tiempo de construcción de la imagen. En el ejemplo particular copia todos los archivos en el directorio de construcción al directorio definido en el WORKDIR.
CMD: Define que script o binario, con sus respectivos parámetros, ejecutan la aplicación al inicio del contenedor.
EXPOSE: Expone el puerto de red, en el que se publicara la aplicación.

Las imágenes son un componente crítico y susceptible a introducir vulnerabilidades o incrementar innecesariamente la superficie de ataque. Por ello debemos sólo incluir componentes que sean estrictamente necesarios.

3.3 Kubernetes

3.3.1 Conceptos básicos

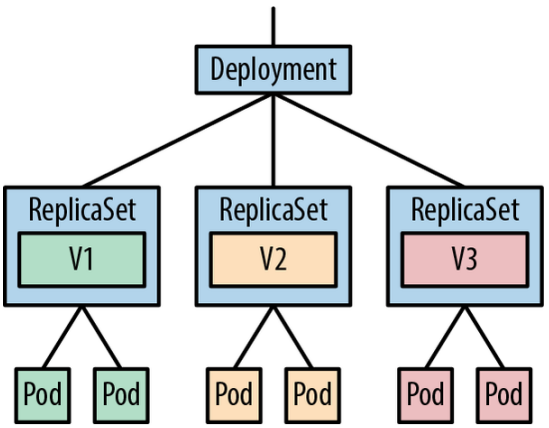
Kubernetes (k8s) es el sistema operativo para el mundo de las aplicaciones nativas en la nube, provee una plataforma escalable y confiable para la ejecución de contenedores. [25]

3.3.2 Componentes y arquitectura de un clúster de Kubernetes

3.3.2.1 Objetos en Kubernetes

La configuración de k8s se almacena por medio de objetos (entidades).Cada objeto tiene su definición, la cual será analizada por kubernetes e intentará llegar al estado deseado. El objeto fundamental en k8s es el Pod, en el siguiente cuadro se detalla los objetos más importantes de k8s.

Nombre	Descripción
POD	Pods son la mínima unidad desplegable que puede crear y manejar k8s. Un Pod es un conjunto de uno o más contenedores, que comparten almacenamiento y recursos de red. Los contenedores por sí solos, no pueden reemplazar a los pods, ya que un pods es un conjunto de uno o más contenedores que comparten recursos computacionales, por lo tanto deben ser ubicados en el mismo nodo, para poder compartir los recursos. [26] Por ejemplo , un Pod de una aplicación PHP, requiere de al menos dos contenedores. Uno para ejecutar PHP y otro para ejecutar el servidor web. Ya que los dos contenedores necesitan del mismo código fuente, almacenado localmente, deben en conjunto formar parte de un POD. K8s permite ejecutar copias exactas de un Pod, estas copias son llamadas réplicas, y serán muy útiles para lograr una alta disponibilidad de los servicios.
Deployment	Un Deployment en k8s es un objeto de recursos, declarativo, que permite declarar múltiples configuraciones para las aplicaciones tales como, las imágenes de los contenedores, el número de réplicas,etc. Los deployment son un conjunto de otro objeto llamado replicaSet. Kubernetes por medio del controlador analizará en bucle el estado del recurso, y llevará al objeto a su estado deseado. Los deployments, también ofrecen la posibilidad de especificar a k8s la estrategía de despliegue deseada.

	En resumen un deployment es un recurso de kubernetes de alto nivel que permite administrar Pods de la siguiente manera. • Despliegue • Actualización • Ubicación en los nodos • Reiniciar
ReplicaSet	Los deployments no manejan los Pods directamente, sin embargo los Replicaset, si lo hacen. Por lo antes mencionado, los Deployment manejan replicaSet, y los replicaSet manejan Pods. Los ReplicaSet son responsable de un grupo identico de Pods, los pods son identificados con un replica set por el campo “metadata.ownerReferences”. Es recomendable utilizar Deployments, y que Kubernetes administre los replicaSet. Una vez iniciada una actualización de un Deployment, el controlador de deployment, creará un nuevo replicaSet y terminará el viejo ReplicaSet cuando el despliegue haya concluido exitosamente.  Imagen Nº15: Esquema jerárquico entre los objetos y sus relaciones.
Servicios	Un Pod dispone de una dirección ip única dentro de la red interna. y el ciclo de vida de un pod se extiende hasta la terminación del mismo. La dirección IP no es estática o previamente conocida. Los servicios nos permiten establecer una dirección IP y un dominio de nombres único para referirse a un conjunto de pods idénticos identificados por una etiqueta. De esta manera el servicio permitirá distribuir las peticiones a diferentes pods ubicados en diferentes nodos dentro del clúster. Un servicio es un equivalente a un balanceador de carga o un proxy, dirigiendo el tráfico a los pods por medio de una única dirección IP.

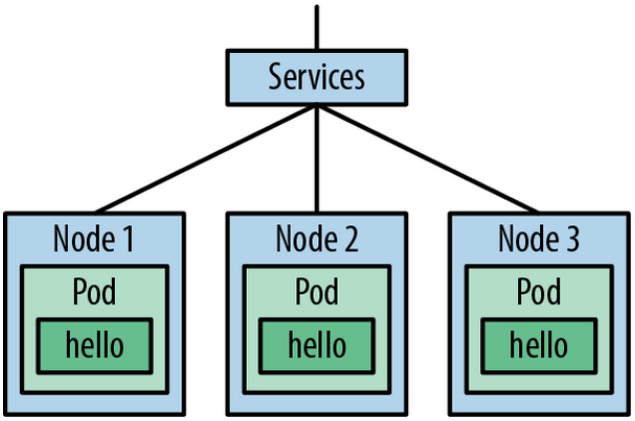


Imagen Nº 16 : Servicio provee un punto de ingreso para los pods distribuidos en los nodos.

Tabla Nº 2 : Objetos de kubernetes

3.3.2.2 Control Plane

Es el cerebro del cluster, también conocido como master , y es el componente principal encargado de llevar a cabo las tareas de k8s, tareas como

- Planificar la ejecución de pods en los nodos
- Activar la interfaz de comunicación del cluster (api)
- Administrar los servicios
- etc

Normalmente el Control Plane se despliega en una o un conjunto de máquinas miembros del clúster. Un mayor número de máquinas garantiza una mejor alta disponibilidad en caso de fallos, sin embargo no garantiza un mejor rendimiento.

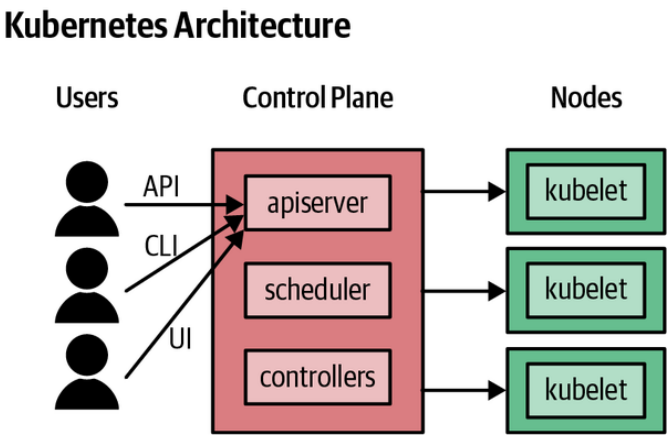


Imagen Nº 17: Esquema despliegue del control plane y los nodos

3.3.2.3 Componentes de software (master o Control Plane)

componentes	descripción
kube-apiserver	Expone y maneja las peticiones a la API
etcd	Base de datos donde se almacena toda la configuración de k8s.
kube-scheduler	Planifica la ejecución de los pods, determina en qué nodo ejecutarlo
kube-controller-manager	Responsable de controlar el estado de los componentes del cluster, por medio de cluster api, y verificar que su estado sea el deseado. Ej controlar que el número de réplicas de un pod sea correcto, si no fuera correcto, toma acciones correctivas. [27]
cloud-controller-manager	Responsable de enlazar el cluster con la API del proveedor de nube, separa los componentes que interactúan con la nube con lo que no. [28]

Tabla Nº 3 : Componentes del control plane

Las cargas de trabajo de usuarios no deben ejecutarse en los nodos master o Control Plane

3.3.2.4 Nodos

componentes	descripción
kubelet	Responsable manejar la ejecución de los contenedores en el nodo y monitorear su estado
kube-proxy	componente de red que se encarga de enrutar las peticiones entre los pods en diferentes nodos o los pods y el exterior.
container runtime	software responsable de ejecutar contenedores en los nodos. [29] Desde k8s 1.24 es oficialmente removido dockershim como opción por defecto
pods	Los pods son la unidad desplegable computacional más pequeña que puedes crear o administrar en k8s. [30]

Tabla Nº 4 : Componentes en los nodos

3.3.2.5 Capacidad y Alta disponibilidad

La capacidad de un cluster está determinada, principalmente, por la cantidad y tipo de nodos que forman el clúster. Cuanto mayor número de nodos, podremos iniciar

un número mayor de pods. Sin embargo, no son los únicos indicadores a tener en cuenta. Por ejemplo, ¿ cuál es la cantidad máxima de POD que puede soportar un nodo, o la cantidad de direcciones de red disponibles por nodo, etc. [31]

Los nodos tanto del Control Plane como de los worker son componentes computacionales propensos a fallar en algún momento indeterminado de tiempo, por lo que k8s requiere un número de nodos (recursos) que puedan tomar la carga de cualquier nodo en caso de fallas.

La alta disponibilidad de nuestras cargas de trabajo a nivel de pods depende de la cantidad de réplicas o copias del pod que se ejecuten al mismo tiempo. Si un nodo falla, por efecto de la falla, todos los pods del nodo mueren o son terminados. K8s es capaz de encaminar las peticiones a otros pods en estado saludable en el cluster.

3.3.2.6 Administración de kubernetes

La administración de kubernetes no es una tarea simple, y el despliegue de un cluster con configuraciones por defecto no es una opción deseable para ninguna organización. Día a día se desarrollan o se mejoran herramientas que hacen los clusters de k8s más robustos y tolerantes a fallos, incluso mucho más fáciles de desplegar.

La comunidad de código abierto y los proveedores de nube son conscientes de esta problemática, cada día se desarrollan nuevas aplicaciones y distribuciones de kubernetes.

3.3.2.7 Distribuciones de K8s

Debido a que k8s es un proyecto de código abierto, como lo son GNU y linux, se adopta el mismo término, muy utilizado , para identificar empaquetados realizados por diferentes proyectos. Por ello podemos afirmar que es un paquete de software que provee versiones precompiladas de k8s. La mayoría de las distros disponen de herramientas de instalación, haciendo el proceso mucho más simple. [32]

Entre los proyectos más famosos podemos encontrar

- EKS
- OCP
- Rancher
- GKE

3.3.2.8 K8s como servicio administrado

K8s como servicio administrado facilita la administración y configuración de clusters significativamente. En especial en lo relacionado al Control Plane. Aunque no podemos evitar las tareas administrativas al 100 %. En pocas palabras, pagamos a terceros (Amazon, Microsoft o Google) para que desplieguen y administren ciertos componentes por nosotros. [33]

3.3.2.8.1 Elastic container service (ECS) Amazon

Amazon inicialmente apostó por su servicio propio para administración y orquestación de contenedores llamado ECS. El servicio funciona correctamente y muchas organizaciones adoptaron esta tecnología para hospedar y administrar sus soluciones basadas en contenedores. Sin embargo ECS, no ha evolucionado como ķ8s , por lo tanto no es tan poderoso o flexible.

3.3.2.8.2 Elastic kubernetes service (EKS) Amazon

Amazon ha apostado por Kubernetes, y ha lanzado el servicio administrado de kubernetes, y hoy en día es uno de los servicios más utilizados en la nube. EKS es un servicio de k8s certificado, que simplifica el proceso de construir , securizar, administrar y mantener clusters de k8s. EKS nos permite integrar servicios fundacionales de AWS como CloudWatch, Grupos de autoescalamiento, IAM , Balanceadores de carga, etc. Con kubernetes. [34]

¿Qué puedo hacer con EKS ?

- Desplegar cluster
- Configuración del cluster
- Despliegue de apps
- Autoescalamiento del cluster
- Monitoreo
- Control de acceso

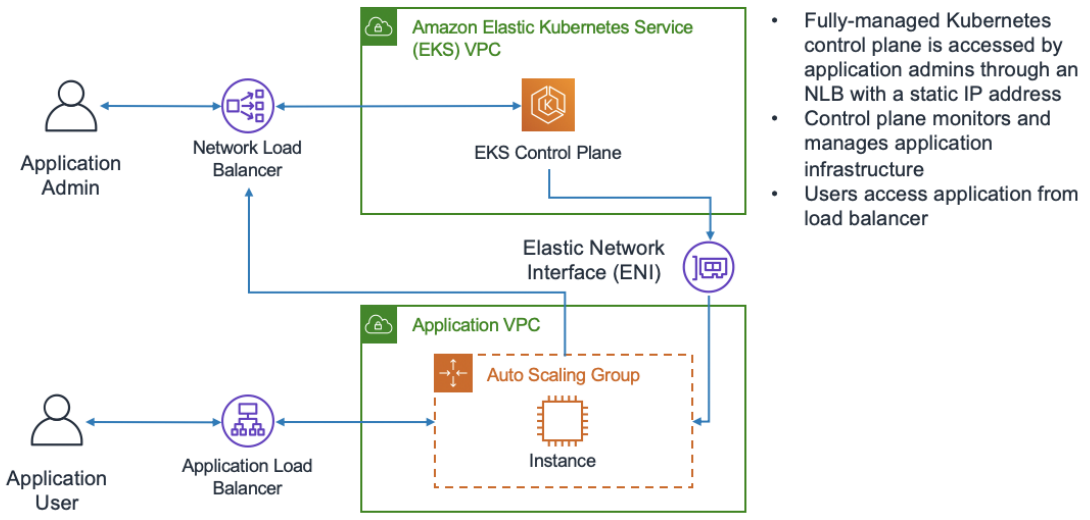


Imagen Nº 18: Esquema básico de un cluster de EKS

EKS se encarga de mantener el Control Plane de cluster, por lo que no debemos preocuparnos por sus actualizaciones o su alta disponibilidad. Es responsabilidad de AWS. Sin embargo, de acuerdo al modelo de responsabilidad compartida de Amazon web services, dependiendo del tipo de configuración elegida, la responsabilidad del usuario se incrementa tanto como quieras personalizar la

configuración del clúster. Por ejemplo sí utilizamos fargate, podremos quitar de nuestro alcance el mantenimiento de los workers o nodos, donde se ejecutan los pods del usuario.

3.4 Terraform & Terraform Cloud

El desarrollo de software no se completa cuando el código funciona en nuestra computadora, tampoco cuando las pruebas de integración y la revisión de pares son superadas. La entrega del software se termina cuando es puesto en producción y los usuarios finales pueden utilizarlos. [35]. Para poder llevar nuestro software a los usuarios finales debemos disponer de una infraestructura adecuada para ejecutar el software.

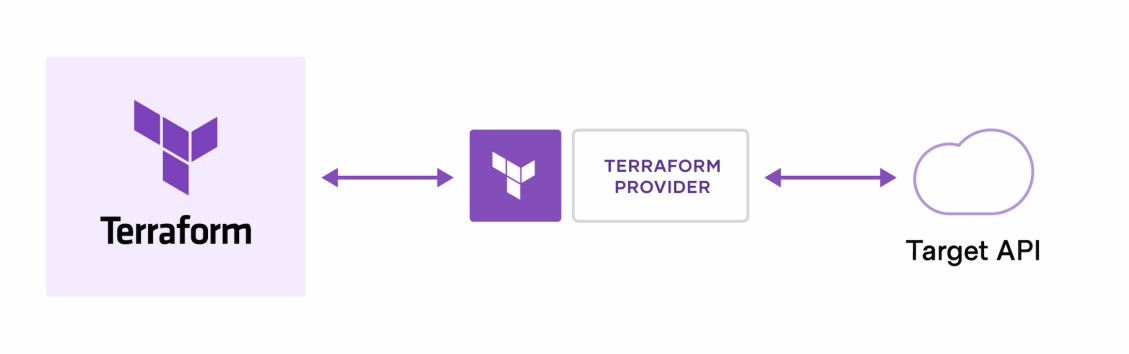


Imagen Nº 19: Terraform se comunica con la API del proveedor

Terraform es una herramienta de código abierto, escrita en Go, que permite la definición de infraestructuras como código , con casi 40 mil estrellas en Github, creado por la empresa Hashicorp es la solución más usada en la actualidad de acuerdo a la estadísticas de usuario / colaboradores en GitHub.

Para la descripción de la infraestructura, terraform provee un lenguaje exclusivo de Hashicorp, el propósito principal del lenguaje es declarar recursos, que representan objetos de infraestructura. [36]

El proyecto de código abierto de terraform nos provee un binario que permite hacer todo el trabajo de desplegar los objetos de infraestructura en la nube o on premise, al mismo tiempo guarda el estado final en un archivo llamado “estado”. Por lo tanto el estado es la fuente de la verdad para referenciar la infraestructura ya desplegada. Cuando nuevos cambios van a ser introducidos, terraform comparará el estado objetivo con el estado actual, y aplicará las diferencias.

Gracias a la naturaleza del proyecto terraform (código abierto), hashicorp y la comunidad han escriben y mejoran diariamente miles de proveedores, por lo que esta solución , al día de hoy, no solo está limitada a los grandes proveedores. [37]

El flujo de trabajo del núcleo de terraform es muy simple, dispone de 3 etapas muy bien diferenciadas.

1. Escritura / Write: Etapa en la que escribimos la configuración a desplegar.
2. Plan: Terraform compara la infraestructura existente en el proveedor con la especificada en el fichero de estado. Las diferencias son mostradas por terraform como un plan. En esta etapa podremos comprender qué cambios se harán efectivos
3. Aplicar cambios: Terraform no aplicará los cambios determinados en la etapa anterior. Se debe proveer una orden explícita de confirmación para que terraform comience a transicionar desde la infraestructura existente a la descrita en el código fuente.

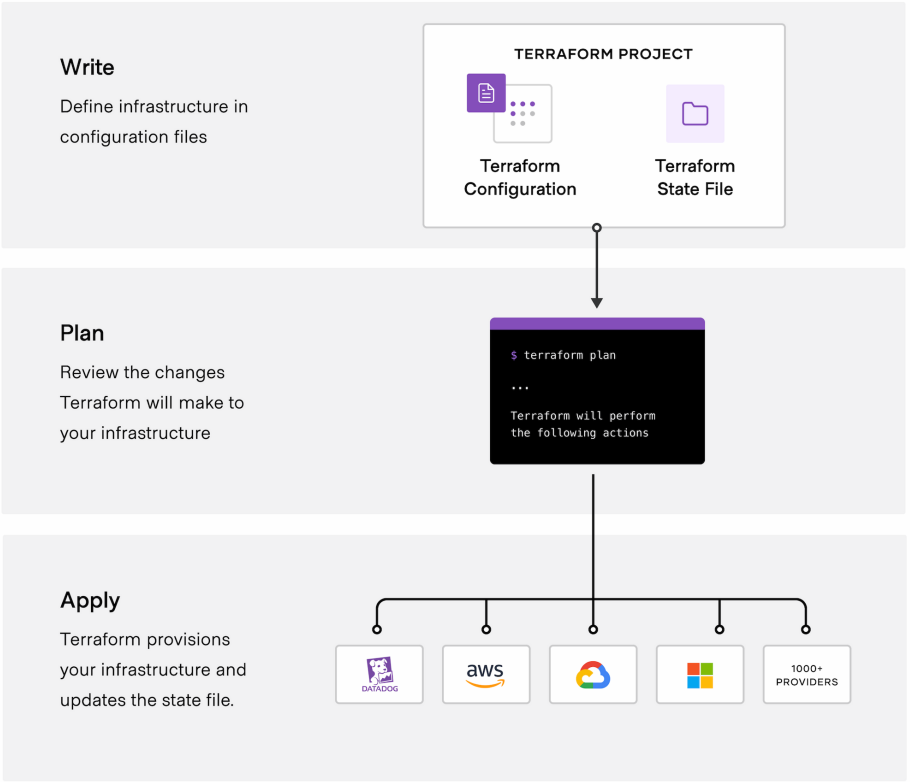


Imagen Nº 20: Etapas

El estado es por defecto almacenado localmente en un fichero llamado terraform.tfstate. Para una persona sola que trabaja en un proyecto no plantea grandes desafíos de usabilidad, sin embargo, plantea grandes preocupaciones desde la mirada de la seguridad. El contenido del tfstate puede contener información sensible almacenada. Los proyectos de software actualmente no son realizados por una persona, sino por un grupo de personas. Por lo tanto el uso de un fichero de estado local es totalmente inviable. [38]

Es evidente que la administración del estado de terraform, no es una tarea trivial, y sugiere los siguientes desafíos.

- El fichero de estado, sólo puede ser accedido por personas autorizadas.
- El fichero debe ser almacenado y transmitido de manera encriptada.
- El fichero sólo puede ser exclusivamente modificado por el binario de terrafor. Edición manual o por software de tercera parte se encuentra totalmente prohibido.
- Se deben guardar copias de seguridad de los diferentes estados. Volver a un estado anterior la infraestructura es una tarea no muy común, pero necesaria.

Hashicorp incorpora una solución cloud , que nos permite solucionar todos estos desafíos. La solución puede usarse de manera gratuita, incluyendo limitaciones de funcionalidades, como número de usuarios. terraform cloud ayuda a equipos con excelente funcionalidades, como las siguientes. [39]

- Datos en reposo y en movimiento encriptados.
- Posibilidad de manejar secretos.
- Se integra con Github y otros controladores de versiones, para poder hacer un control de la transición de código entre ramas. Incluyendo revisión por otros miembros del equipo y ejecución de planes especulativos, como parte del pipeline de infraestructura como código. Un plan especulativo simulará un plan tradicional, en busca de errores, en caso de fallas, no podrá transicionar el código hasta que su solución se haga efectiva.
- Repositorio privado de módulos.
- Políticas de validación para evitar ciertos escenarios.

3.5 Kubescape

Kubescape es un proyecto de código abierto, parte de la Cloud Native Foundation, que propone una plataforma que ayuda a asegurar las diferentes etapas del desarrollo de software e infraestructura. [40]

El proyecto se adapta para ser ejecutado en las etapas de desarrollo, antes de hacer una fusión de ramas, en pipelines, puede ejecutarse como una github action. Incluso el desarrollador puede utilizarlo de manera local mediante su CLI o extensión para Visual studio code o Lens. Por otro lado también puede ejecutarse en el cluster y reportar los resultados de escaneo a la plataforma online.

La plataforma online es servicio SaaS, que puede ser utilizado de manera gratuita con grandes limitaciones. Disponen de versiones pagas de tipo cloud o on premise.

Características

- Puede ser ejecutado en casi todas las etapas del proceso de desarrollo de software.
- Escaneo de vulnerabilidades.
- Visualización de los recursos RBAC de kubernetes y sus interrelaciones.
- Escaneo de registros privados de imágenes o helm charts.
- Ejecutar escaneo benchmark compliance de acuerdo a diferentes framework tales como los de NSA, MITRE, entre muchos otros.
- integración con servicios terceros como slack , jira, entre otros.

3.6 Calico

Calico es una solución de redes y seguridad que permite comunicar cargas de trabajo nativas de k8s como , así también las que no son de kubernetes de manera segura y correcta. [41]

Calico consiste en proveer una red segura de comunicación, y permite la posibilidad de crear política de red para asegurar microservicios o aplicaciones.

3.7 Kyverno

Kyverno es un motor de políticas de seguridad en los clusters de k8s, permite validaciones y protección a nivel de ejecución. Funciona a nivel del controlador de admisión de kubernetes. [42]

3.7.1 Características

- Las políticas son nativas del cloud, puede ser almacenada como objetos de kubernetes.
- Validación: Verifica si algún recurso de kubernetes cumple con los requisitos de la regla. Ej . No permitir ejecutar contenedores con imágenes de terceros.
- Mutar: Si algún recurso no cumple con los requisitos de la política, puede modificarlo de acuerdo a la política. Ej Asegurar que todos los pods tengan resource límite asignado. básicamente, si el pod no tuviera declarado los límites de recursos, kyverno, se los añadiría.
- Generación: Permite la generación de de nuevos recursos de k8s de acuerdo a la definición de la política de seguridad.
- Verificar el contenido de la metadata de las imágenes: Permite o deniega su ejecución de acuerdo a información de la imágenes. Ej verificar una firma digital de la imagen.

```
apiVersion: kyverno.io/v1
kind: ClusterPolicy
metadata:
  name: disallow-helm-tiller
  annotations:
```

```
    policies.kyverno.io/title: Disallow Helm Tiller
    policies.kyverno.io/category: Sample
    policies.kyverno.io/minversion: 1.6.0
    policies.kyverno.io/severity: medium
    policies.kyverno.io/subject: Pod
    policies.kyverno.io/description: >-
      Tiller, found in Helm v2, has known security challenges. It requires administrative
privileges and acts as a shared
      resource accessible to any authenticated user. Tiller can lead to privilege escalation as
      restricted users can impact other users. It is recommend to use Helm v3+ which does not
contain
      Tiller for these reasons. This policy validates that there is not an image
      containing the name `tiller`.
spec:
  validationFailureAction: audit
  background: true
  rules:
    - name: validate-helm-tiller
      match:
        any:
        - resources:
            kinds:
              - Pod
      validate:
        message: "Helm Tiller is not allowed"
        pattern:
          spec:
            containers:
              - name: "*"
                image: "!*tiller*"
```

3.8 KubeArmor

KubeArmor es un sistema de protección en tiempo de ejecución, que permite validar comportamientos tales como ejecución de procesos, acceso a ficheros y operaciones de red a nivel de contenedores , pods, máquinas virtuales.

A diferencia de Kyverno, visto en el apartado anterior, los niveles de protección son diferentes. Kyverno permite validar manifiesto kubernetes previos a su ejecución, sin embargo KubeArmor puede validar operaciones a más bajo nivel.

3.9 ArgoCD

ArgoCD es una herramienta nativa de kubernetes, que permite declaración, GitOps y entrega continua de nuestra aplicaciones o infraestructura.

ArgoCD utiliza el patrón GitOps, con repositorios git como la fuente de la verdad para el estado de las aplicaciones.

3.9.1 Arquitectura

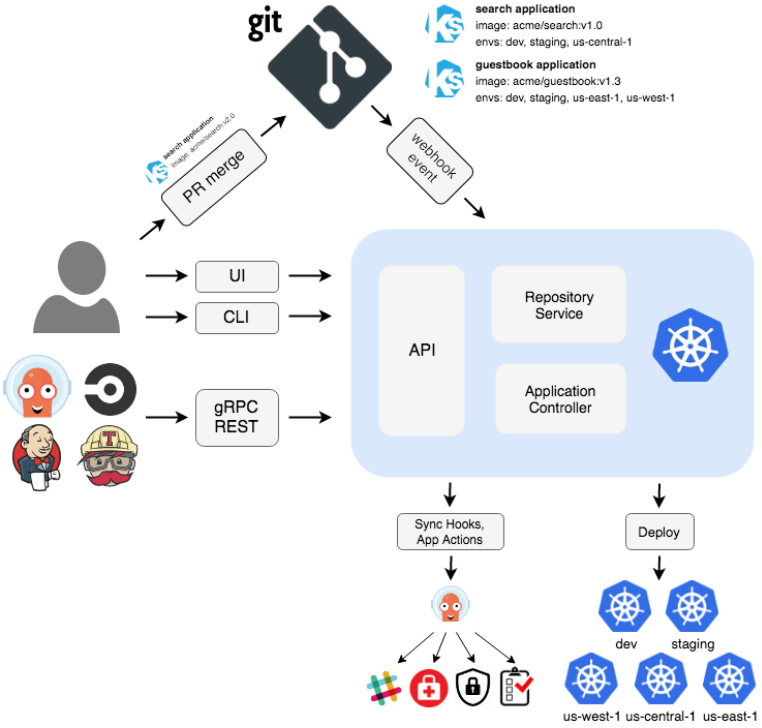


Imagen Nº 21: Arquitectura de ArgoCD

ArgoCD fue implementado como un controlador de kubernetes, el cual continuamente revisa el estado de las aplicaciones ejecutándose y compara su estado actual, con el estado propuesto por el código fuente en Git. ArgoCD reporta las diferencias, a espera de una intervención manual del usuario para modificar el estado de la aplicación o puede proceder a la actualización inmediatamente sin ninguna intervención manual.

ArgoCD puede desplegar aplicaciones en diferentes clusters o simplemente en el mismo cluster donde vive.

3.9.2 Argo y la seguridad

En versiones estable, dispone de características de seguridad muy importantes como: [43]

- Autenticación
- Autorización

- Comunicación segura en la red sobre TLS
- Accesos basado en roles
- Registros de auditorías

Pocos días antes de la entrega del presente trabajo, Argo anunció que trabaja en mejorar su herramienta, para incluir nativamente un mejor control de la seguridad de la cadena de suministro de artefactos de software. Por ejemplo confirmar la autoría y que no fue modificado por terceros un artefacto de software tales como imágenes o helm charts. [44]

3.9.3 Adopción de ArgoCD

ArgoCD es un proyecto de código abierto con un desarrollo muy activo. En Github podemos observar , que activamente los problemas por los usuarios son solucionados y despliegan parches y nuevas versiones muy frecuentemente. CNCF ha catalogado este proyecto con el estado de Graduado, por lo cual es un proyecto listo para producción.

4. Seguridad Nativa en la nube

4.1 Seguridad en capas

Podemos pensar la seguridad en capas, apoyados en el enfoque de defensa en profundidad ampliamente usado en la seguridad de ordenadores. La defensa en profundidad (DiD) es una estrategia de ciberseguridad que utiliza múltiples productos o prácticas de seguridad para proteger el cluster. [45] El modelo es propuesto por la documentación oficial de Kubernetes. [46]

El modelo multi capas sugiere las siguientes, las 4 C

- Código
- Contenedores
- Cluster
- Cloud

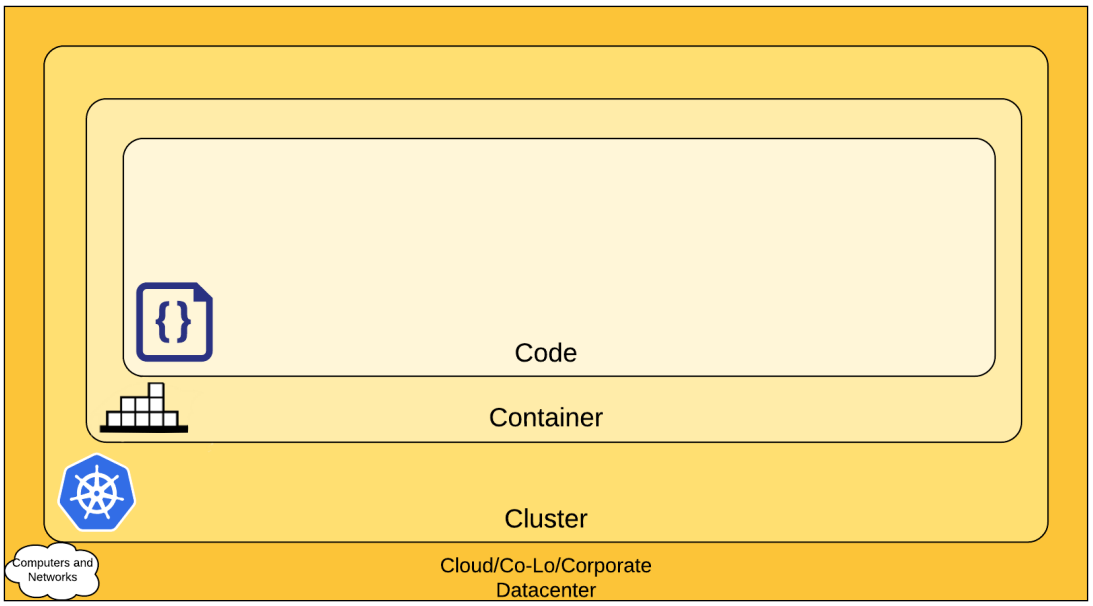


Imagen Nº 22 : Las 4 C de la seguridad en los entorno nativos en la nube

La protección en capas permite analizar y proponer medidas de seguridad en diferentes capas que permitan evitar/ dificultar el acceso a la información de los sistemas.

4.1.1 Proveedor de Nube (Cloud)

Los proveedores de nube, disponen de extensa documentación con buenas prácticas de configuración. Adicionalmente los proveedores de nube definen un acuerdo de responsabilidad compartida. El concepto fue tratado anteriormente en el presente trabajo. Nos limitaremos a Amazon web services y su servicio de cluster kubernetes administrados, también conocido cómo EKS

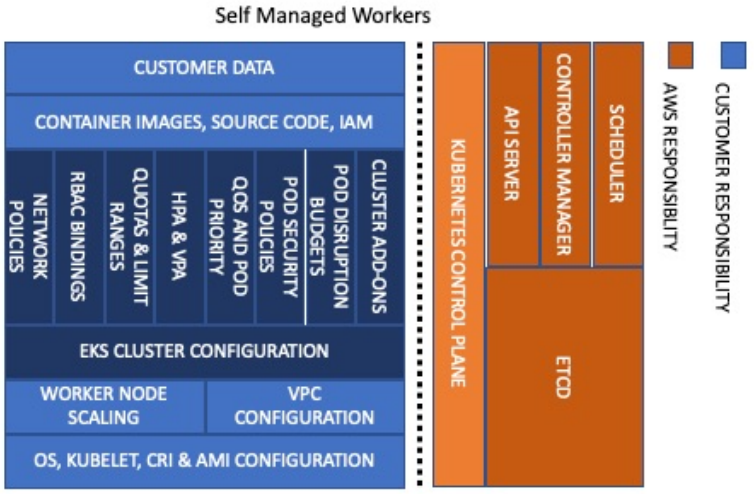


Imagen Nº 23: Distribución de la responsabilidad compartida en EKS , los bloques en azul, son exclusiva responsabilidad del usuario.

En la opción descrita anteriormente, se observa que los nodos son entera responsabilidad del usuario del proveedor de nube. Sin embargo, AWS, provee una alternativa, en la que, AWS, toma responsabilidad de configuración y despliegue de nodos.

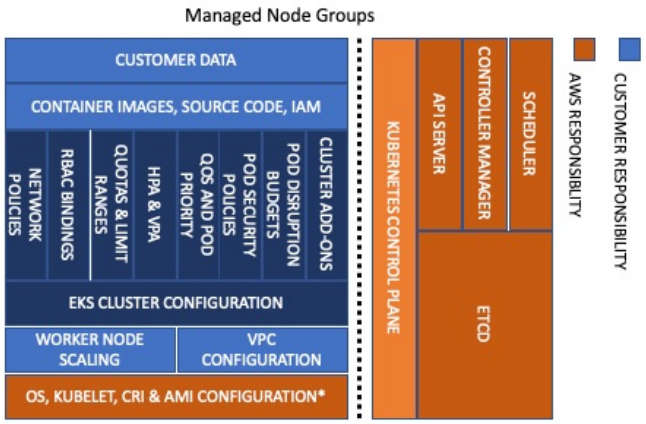


Imagen Nº 24 : La configuración de OS y k8s en los nodos pasa a formar parte de la responsabilidad de AWS

Aunque contratemos un servicio administrado de K8s por nuestro proveedor de nube, existen componentes que deben ser configurados por el usuario, y el usuario es exclusivo responsable de su correcta configuración.

4.1.2 Seguridad infraestructura

La infraestructura de red y máquinas virtuales es un componente muy importante en cualquier organización. Por ello debemos disponer de una correcta configuración. A continuación se exponen algunas recomendaciones para asegurar nuestro cluster.

- Control de acceso al API server /Control Plane: El acceso al Control Plane o la API de kubernetes no debe ser expuesto públicamente, y sólo permitir el ingreso a segmentos de red autorizados
- Control acceso de red nodos: Los nodos deben tener acceso desde el Control Plane en ciertos puertos. También requieren comunicación entre un balanceador de carga y el puerto de red en el nodo que expone un servicio.
- Autorización: La API debe permitir validar usuarios y verificar que el usuario solo pueda realizar acciones, para las cuales fue autorizado. Una asignación correcta de permisos debe seguir el principio de mínimo privilegio. Por ello la asignación de permiso debe ser granular y solo autorizar lo mínimo requerido por cada usuario.

4.1.2.1 Seguridad de la infraestructura en AWS/EKS

Amazon web service publicó el documento de referencia “Infrastructure security in Amazon EKS.” [47]

4.1.2.1.1 Comunicación segura del lado de cliente

Los clientes (software) que interactúan con la API de amazon EKS mediante llamadas deben soportar TLS 1.2 o superior, como así también la suite de cifrado cómo Diffie-Hellman efímero (DHE) o curva eliptica efimera Diffie-Hellman (ECDHE).

4.1.2.1.2 Métodos de autenticación y autorización

Amazon integra su servicio de identificación y control de acceso (IAM), por lo que podremos comunicarnos con credenciales permanentes o temporarias. Una vez autenticado, k8s utiliza acceso basado en roles para autorizar una acción.

4.1.2.1.3 Redes, subredes y VPC

Amazon recomienda el uso de múltiples subredes distribuidas por diferentes zonas de disponibilidad (AZ) (al menos 2, preferentemente 3). Los nodos del cluster deben ejecutarse en subredes de tipo privada, y los balanceadores de carga en las públicas

El servidor API de k8s por defecto es publicado en internet, por lo que debe restringirse el acceso o hacerlo completamente privado y disponible dentro de la VPC.

4.1.2.1.4 Conexión segura con proveedores externos

Los proveedores externos, despliegan sus infraestructuras fuera de los límites de nuestra red corporativa. Para solucionar este problema tenemos diferentes posibles soluciones

- Si las direcciones IP públicas de los proveedores son estáticas y públicamente conocidas. Es posible permitir cierto tráfico de esas direcciones.
- Algunos proveedores, permiten ejecutar sus servicios dentro de los perímetros de nuestra red privada.
- Agentes, es un software instalado dentro de nuestra red, que tiene como objetivo comunicarse con el proveedor de manera segura.

4.1.3 Código fuente (Code)

El código fuente es uno de los componentes más extensos y susceptibles a incluir fallas de seguridad. Una codificación no segura puede exponer información importante de nuestro clientes o negocio.

Recomendaciones sobre codificación segura

- Información en reposo o en proceso de transmisión debe ser encriptada. Por ejemplo, si disponemos de una base de datos, la misma debe ser almacenada en volúmenes de disco encriptados. Evita que los datos puedan ser leídos por terceros no autorizados . Por otro lado, si la información es transmitida sobre una red de datos, la comunicación debe ser codificada de extremo a extremo. Garantiza que terceras partes no puedan interceptar nuestras transmisiones y acceder a la información.
- Limitada exposición de puertos de red TCP/UDP: Sólo se deben exponer puertos realmente necesarios para el funcionamiento de la aplicación y el clúster.
- Analizar los componentes de terceras partes, ya que las librerías externas podrían incluir vulnerabilidades de seguridad. Además se debe considerar actualizar a versiones más recientes, ya que muchos problemas de seguridad fueron resueltos en nuevos paquetes.
- Análisis estático de código: Analiza nuestro código fuente en busca de patrones que puedan indicar un problema de seguridad.
- Pruebas de ataques a nuestros aplicativos o infraestructura: Requiere de un proceso de integración continua, que es una práctica, donde una persona integra su trabajo. y cada integración es verificada por un proceso de compilación y pruebas (Martin Fowler, 2006)
- Toda la configuración de k8s debe ser almacenada en sistemas de control de versiones cómo GIT, además se debe obligar la revisión de código por parte de un o varios colegas, antes de actualizar la rama principal. [48]

4.1.3.1 Administración de secretos en el código fuente

La premisa detrás de GitOps es el uso de git como fuente de la verdad para la infraestructura y la configuración de las aplicaciones, toma ventaja del flujo de trabajo de Git.

Almacenar información confidencial en un repositorio Git representa una vulnerabilidad de seguridad , por lo que no puede ser permitido. Incluso cuando el repositorio git es considerado privado o de audiencia limitada. Si un secreto fue almacenado en texto plano o alguna codificación fácil de revertir. El secreto es considerado comprometido y se debe revocar inmediatamente. [49]

Una solución muy popular y de uso extendido es almacenar, efectivamente el secreto, en el código dentro del repositorio, pero de forma encriptada.

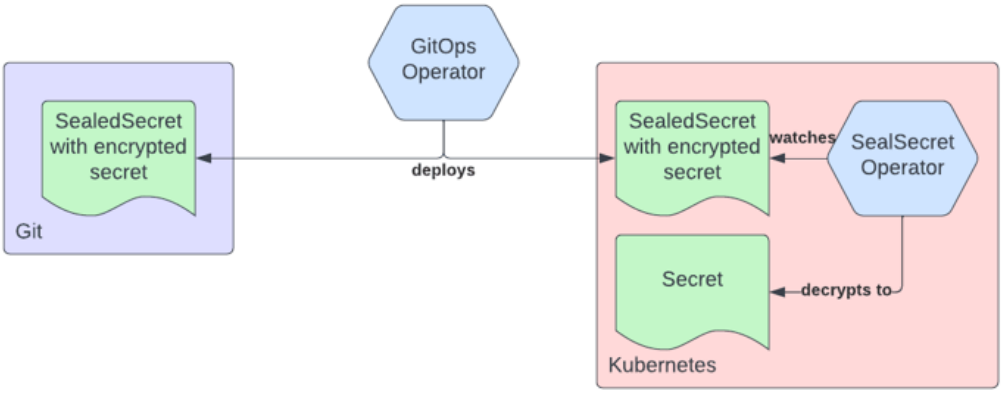


Imagen Nº 25: Secreto encriptado en el repositorio y proceso de desencriptación

4.1.4 Seguridad en contenedores (Contenedores)

Las imágenes de contenedores y las de máquinas virtuales se parecen, pero no son exactamente iguales. Los contenedores son más livianos en número de ficheros (sistema de archivo) y el uso de memoria es menor , por lo tanto, las imágenes, son más rápidas, fáciles de escalar y portar.

Desde el punto de vista de la seguridad, las imágenes deben mantener los paquetes del sistema operativo actualizados, para asegurarnos que no se encuentren comprometidas por vulnerabilidades conocidas y resueltas. Nótese que la misma afirmación aplica para las imágenes de máquinas virtuales.

Las imágenes deben ser creadas y mantenidas bajo la premisa de incluir solamente el software necesario para la ejecución de la aplicación. La inclusión de software no necesario, no solo aumenta su tamaño / costo computacional, sino también extiende la superficie susceptible a incluir fallas de seguridad.

Las imágenes de contenedores, no deben ser estáticas en el tiempo, ya que incluye software de terceras partes, tales como sistema operativo y librerías de ejecución. Por ello es muy recomendable utilizar imágenes bases provenientes de proyectos activos / vivos, que frecuentemente lancen actualizaciones y soluciones a problemas de seguridad de los paquetes que la componen.

Existen herramientas que permiten analizar las imágenes en búsqueda de paquetes con problemas de seguridad. AWS ofrece el servicio Elastic container registry (ECR), que permite almacenar imágenes de contenedores en nuestra infraestructura de manera privada, así mismo revisa las imágenes en búsqueda de vulnerabilidades

Se recomienda el uso de Imágenes firmadas por su creador, y verificación de la firma, antes de la ejecución del contenedor. Esta contramedida, nos permite estar completamente seguro que la imagen no fue modificada maliciosamente. La ejecución de código malicioso, incluido en una imagen, es comparable a la inclusión de un caballo de troya dentro de nuestra infraestructura.

Otra contramedida recomendada en contenedores es evitar el uso de usuarios privilegiados (root), es altamente recomendado crear un usuario con mínimo privilegios dentro del contenedor.

Contenedores con sistema de archivo inmutable, los contenedores sólo deben proveer permisos de escritura a ciertos directorios utilizados por la aplicación, si fuera el caso .

4.1.5 Cluster

Hemos analizado anteriormente cómo un cluster de k8s está conformado, todos sus componentes requieren una correcta configuración, se evita la configuración por defecto. Por otro lado debemos asegurar las aplicaciones o tareas que se ejecutan dentro del cluster. A continuación mencionaremos puntos de preocupación. Es importante resaltar que EKS resuelve automáticamente gran parte de estas recomendaciones, sin intervención del administrador.

4.1.5.1 Recomendaciones que requieren intervención del usuario / administrador

Recomendación	Componente	¿ Por qué ?
Control de acceso a la API de Kubernetes.	Control Plane / API	Es importante definir quienes pueden acceder a la API y qué tareas pueden hacer. Una incorrecta asignación de permisos, podría provocar una brecha de seguridad.
API Autorización	Control Plane / API	Kubernetes nos provee acceso por roles por defecto, pero la configuración de roles y asignación de permisos corre por parte del usuario. Siempre que hablamos de autorización debe primar el principio de mínimo privilegio
Limitar recursos	Worker	K8s provee la ResourceQuota que permiten limitar los recursos que puede tomar los pods en un namespace. Esto evita que una app pueda ocupar todos los recursos del cluster. Existe también la posibilidad de limitar el usuario de recursos computacionales por cada POD
Control de privilegios en la ejecución del contenedor.	Worker	K8s, permite definir contextos de seguridad para los POD, permitiendo explícitamente que privilegios puede obtener.
Prevenir el acceso al metadata endpoint del proveedor de nube	Worker	No es necesario que un POD pueda acceder a esta API, es incluso peligroso, por ello debe bloquearse por medio de políticas de red.
Encriptar secretos al almacenarlos	Control Plane	EKS permite almacenar secretos o configmaps de manera encriptada en la base de datos ETCD

Tabla Nº 5 : Recomendaciones

4.1.5.2 Recomendaciones resueltas por defecto por EKS

Recomendación	Componente	¿ Por qué ?
Uso de TLS en todo el tráfico entrante / saliente de la API	Control Plane / API	La encriptación de las comunicaciones evita que terceras partes puedan acceder a nuestra información de acceso mientras viaja por la red.
API autenticación	Control Plane / API	Evitar que cualquier atacante pueda acceder a la API y ejecutar operaciones es fundamental. Por defecto EKS utiliza IAM para acceder, pero puede utilizarse otros protocolos.
Kubelet API deshabilitar acceso anónimo	Worker	Desactivar acceso anónimo a la API de kubelet
Registro de auditoría	Control Plane	EKS permite la colección y almacenamiento de registros del control plane.
Acceso restringido a ETCD	Control Plane	EKS por defecto restringe el acceso a ETCD y no permite el acceso. Solamente podemos interactuar con etcd por medio de la API de kubernetes.

Tabla Nº 6 : Recomendaciones resueltas por EKS

4.1.5.3 Aislamiento de red.

La segmentación de redes y la securización de la red del Cluster es un concepto central en k8s, la comunicación entre contenedores, Pods, servicios internos y externos debe ser pueda en consideración. Por defecto los recursos de kubernetes no se encuentran aislados y permiten movimientos laterales y escalamientos si un cluster fuera comprometido (NSA,2022)

4.1.5.4 Espacios de nombres (namespace)

Los namespace o espacio de nombre son un recurso de kubernetes que proveen un mecanismo de agrupamiento de recursos en un único clúster. Por defecto no se encuentran aislado, sin embargo con políticas de red y de rbac

4.1.5.5 Políticas` de seguridad de red (network policy)

Las políticas de red controlan el tráfico entre Pods, namespaces, y direcciones IP externas. Por defecto K8s no aplica ninguna política de red, ergo la comunicación puede fluir dentro de la red del cluster sin limitaciones. (NSA,2022)

Recomendaciones del framework NSA

- Elección de un complemento CNI que soporte política de red. Por ejemplo Calico es un CNI opensource que soporta política de red
- La política de red por defecto debe denegar todo el tráfico que ingrese y egrese.
- Declarar politica de red para la comunicación entre pods o namespaces

4.1.5.6 Imágenes base de máquinas worker (AMI)

Las imágenes base de las máquinas worker son un componente muy importante, deben incluir todo el software necesario para poder ejecutar sin problemas k8s. La inclusión de software no relacionado con el objetivo principal no nos beneficia desde el punto de la seguridad. Las distribuciones de linux tradicionales incluyen software que no es necesario. AWS conociendo la falencia de las distribuciones más usadas, decidió lanzar un nuevo proyecto de código abierto llamado Bottleroket diseñada exclusivamente para alojar contenedores. Bottlerocket incluye solo el software esencial para ejecutar los contenedores, lo que mejora el uso de los recursos, reduce las amenazas a la seguridad y reduce los gastos de administración. [50]

4.1.5.7 Registros (logging)

K8s clusters deben enviar todos los registros generados a un sistema de almacenamiento de registros, AWS dispone del servicio CloudWatch Logs, que permite almacenar los registros y luego consultarlos en caso de ser necesario.

La recolección de registros debe contemplar componentes de todos los niveles del cluster, tales cómo el nodo, contenedores, repositorio de imágenes docker, servidor k8s API, incluso de la nube. En el caso de la nube de amazon, dispone de un servicio que guarda todas las interacciones con la API de amazon, llamado CloudTrail.

La recolección de registros es un requisito fundamental para la auditoría de la infraestructura y el cluster. Los registros pueden utilizarse de diferentes maneras. De manera proactiva podemos analizar los registros y por medio de reglas predefinidas detectar anomalías o fallas del sistema. Por otro lado, recolectamos información valiosa para realizar prácticas forenses, en caso que fuera necesario.

5. Resultados

5.1 Detalles del producto

Nombre del producto: EKS security framework

Objetivos:

- Generar una herramienta de infraestructura como código (abierto) que pueda desplegar clusters de kubernetes (eks) de manera automática e incluyendo medidas de seguridad
- Desplegar configuraciones recomendadas de acuerdo las buenas prácticas de seguridad
- Desarrollar casos de usos con la herramienta.

5.2 Distribución componentes

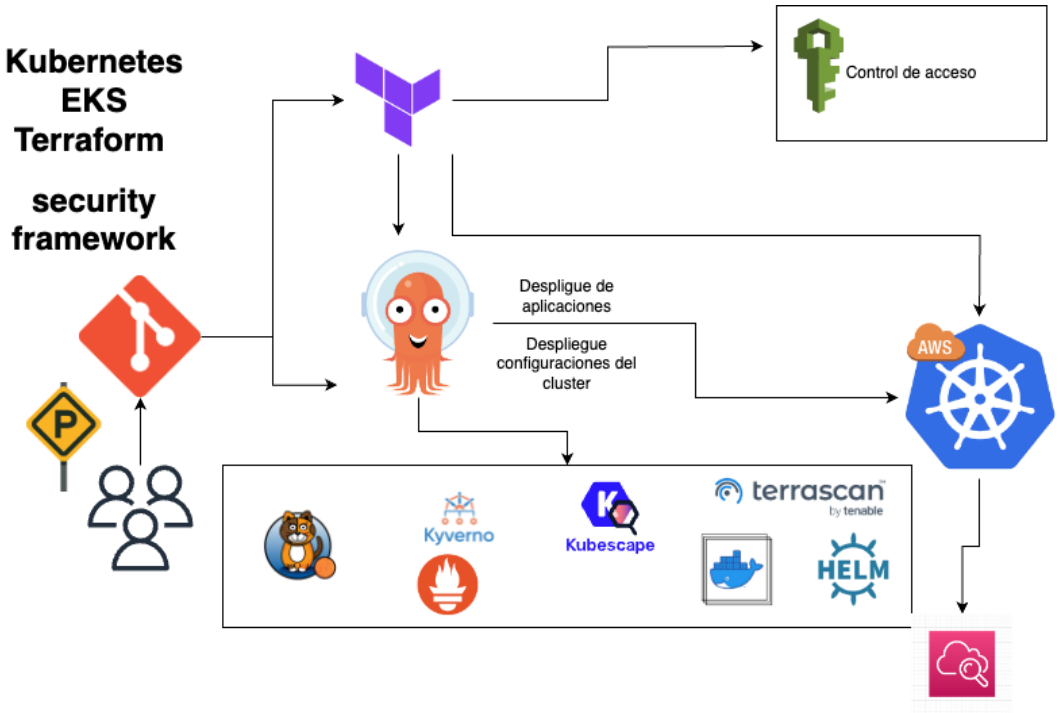


Imagen Nº 26 : Distribución de componentes del producto

5.3 Descripción

Framework, de código abierto , que incluye recomendaciones de cómo desplegar un entorno de kubernetes con énfasis en la seguridad. Se incluyen recomendaciones y buenas prácticas de seguridad en las diferentes capas del modelo de defensa en profundidad de seguridad nativa en la nube, propuesto por los desarrolladores de kubernetes.

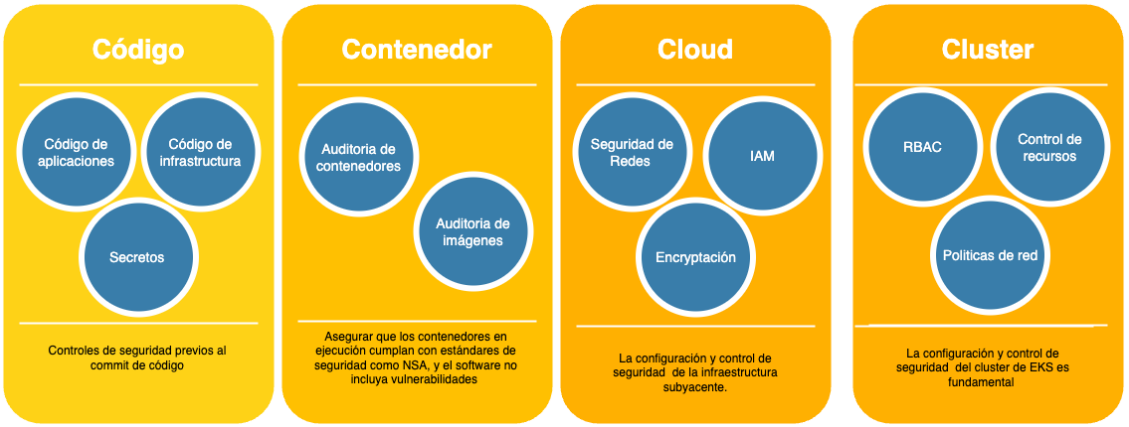


Imagen Nº 27: Características por capa

Además de la presente memoria, se creó un conjunto de repositorios, que juntos nos guiarán en la implementación de nuestro entorno seguro.

Es apropiado aclarar que el framework pretende ser una guía y herramienta de código abierto, que permita al usuario desplegar un entorno con recomendaciones y buenas prácticas de seguridad empresarial en entornos nativos del cloud. Por lo tanto no se debe considerar como una guía definitiva.

Repositorio	Descripción
terraform-aws-eks url: https://github.com/terraform-aws-modules/terraform-aws-eks	Repositorio público, creado por AWS, que nos permite crear clusters EKS con terraform
terraform-aws-eks-security-clusters url:https://github.com/abelnieva/terraform-aws-eks-security-clusters	Módulo de terraform, que funciona como un envoltorio, el cual crea cluster, todo su entorno cloud mínimo y aplica a los clusters una configuración inicial segura básica
eks-security-framework url: https://github.com/abelnieva/eks-security-framework	Repositorio principal para recopilar documentación, casos de usos y configuraciones adicionales que nos ayudarán a entender el uso de los demás repositorios.

eks-security-terraform-workspace url: https://github.com/abelnieva/eks-security-terraform-workspace	Ejemplo de configuración de terraform básica.
eks-security-framework-base url: https://github.com/abelnieva/eks-security-framework-base	Configuración base, que incluye, políticas de seguridad del cluster, y toda configuración global.
eks-security-framework-apps url : https://github.com/abelnieva/eks-security-framework-apps	Repositorio para definir las aplicaciones del negocio a desplegar en el cluster.
eks-security-framework-add-ons url: https://github.com/abelnieva/eks-security-framework-add-ons	Repositorio configuración de aplicaciones para ser instaladas en el cluster. Es un fork de eks-blueprints-add-ons

Tabla Nº 7 : Repositorios del framework

5.4 Características

Las característica desarrollada, para una mejor comprensión, son divididas por capas

5.4.1 Código

- Control de código previo al commit en un repositorio.
- Git Branching y Pull Requests.
- Github actions control de infraestructura (terrascan).
- Análisis estático, vulnerabilidades y buenas prácticas del código.

5.4.2 Contenedor

- Escaneo de imágenes de contenedores, en busca de vulnerabilidades y compliance benchmark framework (kubescape).

5.4.3 Cluster

- Despliegue de cluster de k8s distribución EKS por medio de un módulo terraform.
- Despliegue de aplicaciones con ArgoCD.
- Despliegue de imagen en un repositorio OCI interno (ECR).
- Creación de equipos con limitado acceso por RBAC.
- Colectar y analizar logs de kubernetes para auditoría.
- Manejo de políticas seguridad de admission controller (kyverno).

- Políticas de red en la red del cluster / Aislamiento de comunicación entre micro servicios / namespaces (políticas de seguridad).
- Ejecución de CSI benchmarks para kubernetes.
- Registros de auditorías.
- Despliegue de agentes terraform cloud y kubescape para comunicación segura con proveedores externos.

5.4.4 Cloud

- Control de accesos IAM.
- Configuración de red de AWS.
- Encriptación en reposo.
- registros para auditorías.

5.5 Alcance

El alcance del proyecto incluye desplegar configuraciones de seguridad desde que el desarrollador se encuentra escribiendo la infraestructura o aplicaciones hasta que las aplicaciones son ejecutadas en k8s.

5.6 ¿Por qué EKS ?

De acuerdo al reporte del estado de k8s publicado por Redhat en el año 2023. EKS es la opción más utilizada por los usuarios. [51]

5.7 Criterio de elección de aplicaciones de seguridad

Durante la investigación inicial y detallada, para la realización del trabajo final de máster, se evidenció la existencia de proyectos de código abierto que realizan tareas similares. El criterio de decisión para elegir los productos fueron mayores funcionalidades, comunidad activa. Se prioriza el uso de productos que abarquen más funcionalidades, que tenga una comunidad activa, y un mayor número de usuarios.

5.8 Catálogo de aplicaciones de seguridad

Nombre	Descripción	Fases
Terraform cloud	Plataforma integrada con github que permite administrar los pipeline de infraestructura como código	
terrascan	Analizador de código estático (terraform) para infraestructura como código	Desarrollo Local Pipelines
Precommit hook - terraform	Herramienta de código abierto que permite realizar verificaciones en nuestro código antes del push a un repositorio git	Desarrollo Local Pipelines
KubeArmor	KubeArmor es un herramienta de control a nivel de ejecución que puede restringir la ejecución de procesos. acceso a archivos y operaciones de red tanto en contenedores , como en nodos	Ejecución
Kyverno	Motor de políticas diseñado para kubernetes	Ejecución
Cloudwatch	Servicio de AWS que permite almacenar y consultar registros / métricas generados por él cluster.	Ejecución
Kubescape / Armor	Plataforma open source de apoyo a la seguridad	Multietapas
Calico	CNI de kubernetes con soporte de políticas de seguridad	Ejecución

Tabla Nº 8 : Aplicaciones de seguridad

5.9 Seguridad en Capas Vs Ciclo de desarrollo del software

Basado en la investigación realizada para el desarrollo del trabajo final de máster, la seguridad puede ser aplicada en capas, tal como lo muestra el modelo de defensa en profundidad, sin embargo existe una fuerte relación con el ciclo de vida de desarrollo del software. El flujo de control de la seguridad automatizado debe existir en cada una de las etapas del flujo de trabajo de una empresa, como así también en todas las capas de la infraestructura virtual o física. El componente de integración y despliegue continuo (CI/CD) es el componente fundamental en el desarrollo de software para asegurar detectar fallas o vulnerabilidades en etapas tempranas del

proyecto. Por ello utilizaremos pipelines CI/CD para dotarlos de controles extra de seguridad

5.10 Codificación y pipelines

El framework provee plantillas de repositorios git , para la codificación de infraestructura terraform y de aplicaciones con pipelines y controles previos a un commit de código.

5.10.1 Pre-commit

Controlar la fase previa al commit es de vital importancia porque evita agregar código potencialmente mal formado o vulnerable. Básicamente prevenimos introducir incidencias de seguridad al repositorio [OWASP][1].[52]

Tanto desarrolladores muy experimentados, como novatos son beneficiarios de los controles previos al commit

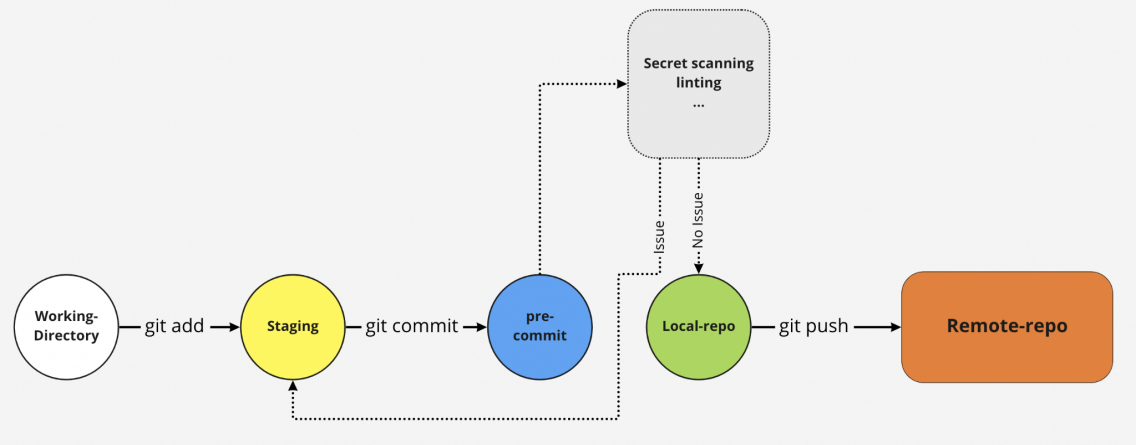


Imagen Nº 28 : Flujo de pre commit

5.10.1.1 Configuración

Por defecto para código terraform, las plantilla proveen los siguientes controles

- Espacios en blancos al final.
- Detecta claves privadas o secretos en código.
- Detecta la exposición de claves de AWS en código.
- Previene la inclusión de código mal formateado o incompleto, para ello ejecuta un linter de código terraform.

[1] OWASP DevSecOps Guideline - v-0.2 [Internet]. OWASP Foundation. [cited 2023 May 14]. Available from: https://owasp.org/www-project-devsecops-guideline/latest/01-Pre-commit

5.10.1.2 Ejemplo de ejecución

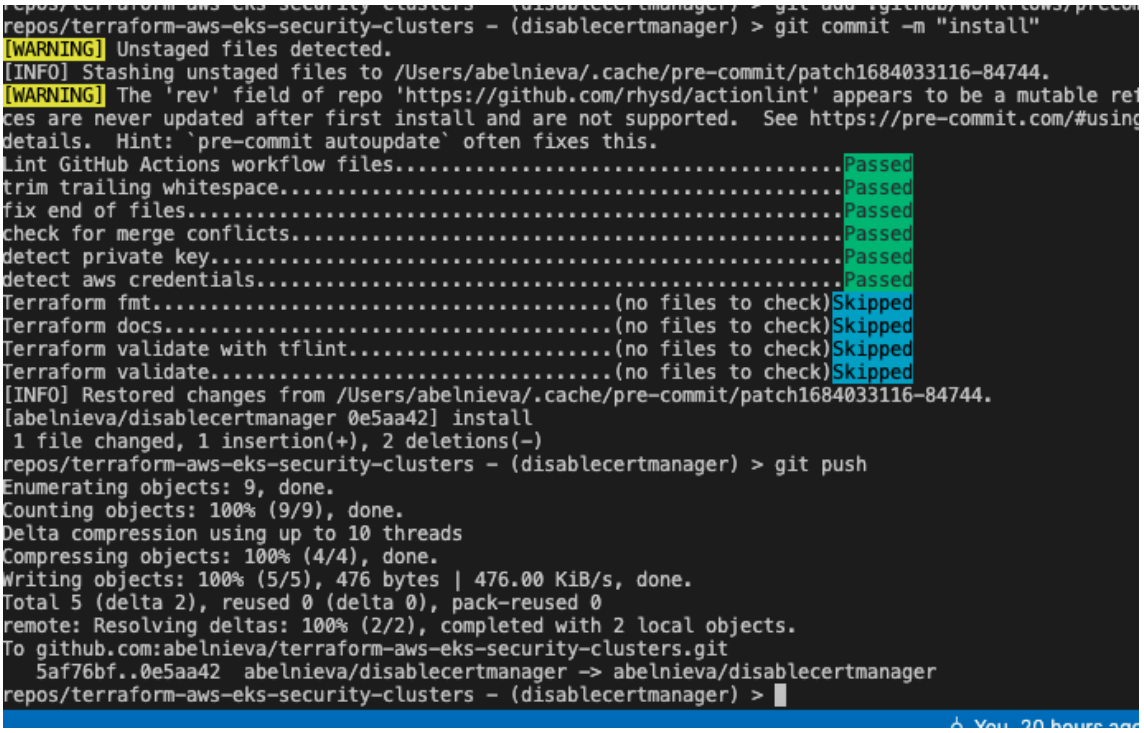


Imagen Nº 29: Salida estándar de pre commit al intentar hacer un commit al repo local.

5.10.2 Kubescape cómo extensión de visual studio code

Visual Studio Code es uno de los IDE más usados por los desarrolladores a nivel mundial, su éxito se debe a su capacidad de extender sus funcionalidades por extensiones desarrolladas por terceras partes. Kubescape no podría ser la excepción, publicó recientemente una extensión que nos realiza sugerencias mientras escribimos nuestro código en lenguajes nativos del cloud. En el repositorio eks-security-framework encontrarás la guía de instalación completa

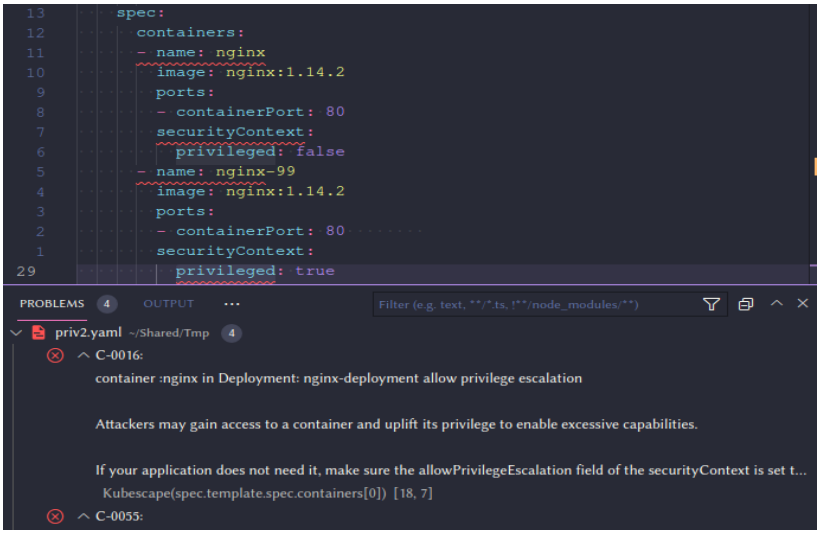


Imagen Nº 30: Detalla posibles problemas de seguridad mientras se codifica.

5.10.3 Seguridad en los pipelines de CI/CD

Existen múltiples herramientas en el mercado para el almacenamiento de código (repos), que actualmente proveen herramientas para la ejecución de pipelines CI/CD sin necesidad de herramientas adicionales de administración, como era el caso de Jenkins. El framework incorpora en sus plantillas Acciones de GitHub.

Para el código terraform se incluye controles similares a los expuesto en la etapa de verificación local (máquina del desarrollador). De acuerdo a la naturaleza del proyecto y lenguaje, podría ser susceptible a incorporar mayores controles.

La plantilla incluye por defecto 3 github actions para terraform

1. Pre-commit: El mismo test ejecutado localmente.
2. Terrascan: Aplicación de código abierto que revisa código terraform en busca de posibles fallas de seguridad o malas configuraciones, nos permite leer el reporte antes de hacer una fusión de ramas
3. Kubescape: Los reportes de análisis de código en la pull request, son enviado a la plataforma ARMO

5.10.4 ¿Por qué duplicar la ejecución de pre-commit?

Desde un punto de vista del cumplimiento obligatorio de ejecución de los controles , solamente desde el pipeline podemos asegurar que solo código que cumple todos los requisitos pueda fusionarse en la rama principal. El ordenador del desarrollador no es algo fácil de controlar.

5.10.5 ¿Qué beneficios provee descubrir problemas en etapas de pre-commit o pipelines ?

Dawson, Maurice y otros en su publicación Integrating Software Assurance into the Software Development Life Cycle afirman basados en un estudio de IBM que un defecto detectado en la etapa de pruebas es 15 veces más caro que si hubiera sido descubierto en etapas de diseño. [53]

5.10.6 Protección de ramas & revisión de pares

En el framework se explica cómo proteger una rama principal de fusiones, sin aprobaciones o los pipelines hubieran fallado. La ausencia de protección de ramas provocaría que las etapas de control en local y pipelines fueran inútiles, debido a que código incorrecto puede formar parte de la rama principal sin ninguna limitación real.

5.11 Contenedores

Los contenedores , los pods , los servicios, los deployments , etc. Son componentes del mundo de kubernetes susceptibles a contener errores de configuración o proveer más permisos de los necesarios. Como contramedida la plataforma ARMO y Kubescape, nos permite explorar la configuración de los objetos de kubernetes y proponer posibles soluciones.

Otro punto importante de validación en los contenedores, son sus imágenes. Al igual que con los contenedores. ARMO y Kubescape analizarán las imágenes, en busca de vulnerabilidades. Tener conocimiento de las vulnerabilidades que afectan a nuestro cluster actualmente, nos ayudará encontrar actualizaciones de nuestras aplicaciones con la vulnerabilidad resuelta.

Un último punto, que fue tratado anteriormente en la memoria, tiene que ver con la generación de imágenes compactas y que solo incluya el software realmente necesario. Para un diseño óptimo de imágenes, se incluye en el framework, un documento tratando al detalle el tema. Claramente no podemos diseñar imágenes creadas por terceras partes, sin embargo, si lo podemos hacer con nuestras propias creaciones.

5.12 Cluster & Cloud

5.12.1 Infraestructura como código

En capítulos anteriores, se ha mencionado los beneficios de tratar la infraestructura como código, por lo tanto continuaremos bajo la premisa de tener toda nuestra configuración en código y dentro de un repositorio de control de versiones. El código es la fuente de la verdad de nuestra infraestructura.

5.12.2 Módulo de Terraform

Para el despliegue automatizado de cluster de tipo EKS, se ha desarrollado un módulo que automatiza , no sólo la creación de un cluster EKS en AWS, sino también características que benefician y facilitan a la creación de entorno nativos en la nube

El repositorio se llama terraform-aws-eks-security-clusters, se encuentra publicado como parte del framework, a continuación un ejemplo de una llamada del módulo.

```
module "cluster_infra" {
  source                                                                    =
"app.terraform.io/security-framework/eks-security-clusters/aws"
 version                                = "0.0.2"
 cluster_name                           = "test-cluster"
 vpc_cidr                               = "10.0.0.0/16"
 cluster_endpoint_public_access         = true
 ecr_repos_list                         = ["testrepo"]
 dev_teams = {
   dev_1_team = {
     users = ["arn:aws:iam::XXXXX:user/user1"] #arns
     namespaces = {
       default = {
         # Provides access to an existing namespace
         create = false
       }
     }
   }
 }
 kubescape_account_id = var.kubescape_account_id //sensible
       repo_apps_url_dev                                                    =
"git@github.com:abelnieva/eks-security-framework-apps.git"
 repo_apps_path_dev       = "teams/dev_1_team/dev"
 dns_zones             = []
 managed_node_groups = {
   node_group_a = {
     min_size = 1

     max_size           = 10
     desired_size       = 3
     launch_template_os = "bottlerocket"
     ami_type           = "BOTTLEROCKET_x86_64"
     instance_types     = ["t3.large"]
     capacity_type      = "SPOT"

     update_config = {
       max_unavailable_percentage = 33 # or set `max_unavailable`
     }

     tags = {
       ExtraTag = "example"
     }
   }
 }
}
```

Imagen Nº 31: Ejemplo de interfaz de parámetros del módulo

5.12.2.1 ¿Qué podemos personalizar ?

Resultado	Variables	Valores ejemplo
Creación de red privada virtual en AWS con subredes distribuidas en zonas de disponibilidad.	vpc_cidr	"10.0.0.0/16"
Posibilidad de restringir el acceso a la API del cluster por medio CIDR	cluster_endpoint_public_access_cidrs	["84.79.52.183/32"]
Creación de repositorios privado de ECR para nuestras imágenes	ecr_repos_list	[“repo1”, “repo2”]
Permite la creación de zonas DNS en route53, y luego asociarla a los deployments del cluster por medio de external-dns	dns_zones	[“midominio.es”]
Permite la instalación y asociación del agente de kubescape en el cluster con la plataforma ARMO	kubescape_account_id	“8s8s8s8s8s8s-s8s8s”
Permite la definición de node groups en el cluster de kubernetes. Un cluster puede tener 1 o más node groups . Esta variable utiliza detrás de escena al modulo https://github.com/terraform-aws-modules/terraform-aws-eks.	managed_node_groups	{ node_group_a = { min_size = 1 max_size = 10 desired_size = 3 launch_template_os = "bottlerocket" ami_type = "BOTTLEROCKET_x86_64" instance_types = ["t3.large"] capacity_type = "SPOT" update_config = { max_unavailable_percentage = 33 # or set

		`max_unavailable`
		}
		
		tags = {
		ExtraTag = "example"
		}
		}
		}

Podemos definir todos los equipos que utilizaran nuestro cluster. El cluster creará roles con accesos limitados de acuerdo a la configuración ingresada. Por detras de escena se utiliza el módulo https://aws-ia.github.io/terraform-aws-eks-blueprints/teams/	dev_teams	{
		dev_1_team = {
		users = ["arn:aws:iam::XXXX:user/user1","arn:aws:iam::XXXX:user/user2"] #arns
		namespaces = {
		default = {
		# Provides access to an existing namespace
		create = false
		}
		}
		}
		}

Por defecto kubernetes guardará registros de los siguientes componentes. Por medio de esta variable, puede ser personalizado. Se recomienda mantener valores por defecto.	cluster_enabled_log_types	["api", "audit", "authenticator", "controllerManager", "scheduler"]
Agregar tags o etiquetas a los recursos en la nube es una muy buena idea, y debería ser obligatorio.	tags	{}
ArgoCD es el responsable de desplegar aplicaciones en el cluster de kubernetes en conjunto con terraform. Por medio de esta variable, podemos determinar la versión del helm chart a instalar .	argocd_version	"5.19.14"
Instala una máquina ec2 con el agente de terraform cloud. Necesario para despliegue de clusters privados con terraform cloud	tfc_agent_token	“Token”

Tabla Nº 9 : Listado de valores soportados de entrada al módulo de terraform

5.12.2.2 ¿ Qué no podemos personalizar ?

- La configuración de red no es muy personalizable, solo podemos introducir el CIDR de red que nos gustaría utilizar, el módulo creará siempre
    - 3 subredes públicas
    - 3 subredes privadas
    - VPC endpoints
    - Configuración de security groups interno permitiendo solo el acceso entre componentes necesario
    - Cloudtrail a s3
- Aplicaciones instalada por defecto
    - Kyverno
    - Calico
    - external dns
    - cluster autoscaler
    - ebs scsi controller
    - fluent bit

5.12.2.3 Si mi organización requiere mayor personalización del módulo ¿Qué hago?

¡Enhorabuena!, tú organización ha encontrado utilidad en Kubernetes, y es esperable que las organizaciones requieran mayor personalización. El framework fue creado como parte de un proyecto final de máster, la naturaleza del proyecto restringe el tiempo de desarrollo de funcionalidades. Por lo cuál invitamos a hacer un fork del módulo y personalizarlo al detalle de acuerdo a vuestras necesidades. Sin embargo se planea mantener el desarrollo del proyecto activo y sumar nuevos colaboradores y funcionalidades.

5.12.3 Despliegue de configuraciones bases/seguridad en el cluster.

Repositorios plantillas:

- eks-security-framework-base

Se ha omitido intencionalmente hablar de un conjunto de variables que incluye el módulo de terraform, que nos permitiran desplegar aplicaciones por medio de ArgoCD en nuestro cluster. Las variables para la configuración base son repo_base_url y repo_base_path. Ambas variables contienen la dirección del repositorio y la ubicación interna. Previamente debemos hacer un fork del repositorio eks-security-framework-base, ya que debe considerarse como una plantilla, a la cual debemos personalizar de acuerdo a vuestras necesidades.

5.12.3.1 Estructura del repositorio

Directorio	Propósito
AdmissionControllerPolicies	políticas de seguridad en formato entendible para kyverno.
NetworkPolicies	Políticas de red globales en formato entendible por Calico
RuntimePolicies	Políticas de runtime en formato kubearmor

Tabla Nº 10 : Directorios del repositorio

5.12.4 Despliegue de aplicaciones propias

Repositorios plantillas:

- eks-security-framework-apps

Al igual que el apartado inmediato anterior, se ha omitido intencionalmente mencionar la existencia de dos variables del módulo de terraform, que nos permiten agregar nuestro módulo de aplicaciones, normalmente un fork de eks-security-framework-apps, o cualquier que el usuario desee que siga las normas de codificación de ArgoCD o Kustomize

La plantilla propone una manera de distribuir las aplicaciones dentro del repositorio, pero el uso de esta plantilla es totalmente opcional. Las variables son repo_apps_path_dev, repo_apps_url_dev, repo_apps_path_prod, repo_apps_url_prod. La única limitación es utilizar código de tipo helm chart

5.12.5 Despliegue de add-ons

Repositorio plantilla:

- eks-security-framework-add-ons

El repositorio de add-ons es un fork del ejemplo propuesto por AWS(https://github.com/aws-samples/eks-blueprints-add-ons). Implementa una posible automatización para el despliegue de aplicaciones con terraform y ArgoCD, utilizando el paradigma de GitOps. En el archivo chart/values.yaml, podemos activar diferentes aplicaciones a ser instalada en el Cluster, también es posible agregar valores de configuración de las aplicaciones. Los valores deben ser soportados por el helm chart.

6. Ejemplo de uso

6.1 Generación entorno de trabajo terraform

Repositorios requeridos:

- terraform-aws-eks-security-clusters
- eks-security-terraform-workspace

Aplicaciones requeridas

- terraform
- terraform cloud
- git
- AWS CLI
- Cuenta de AWS
- Plataforma ARMO

6.1.1 Creación cuenta terraform cloud

URL: https://app.terraform.io/public/signup/account

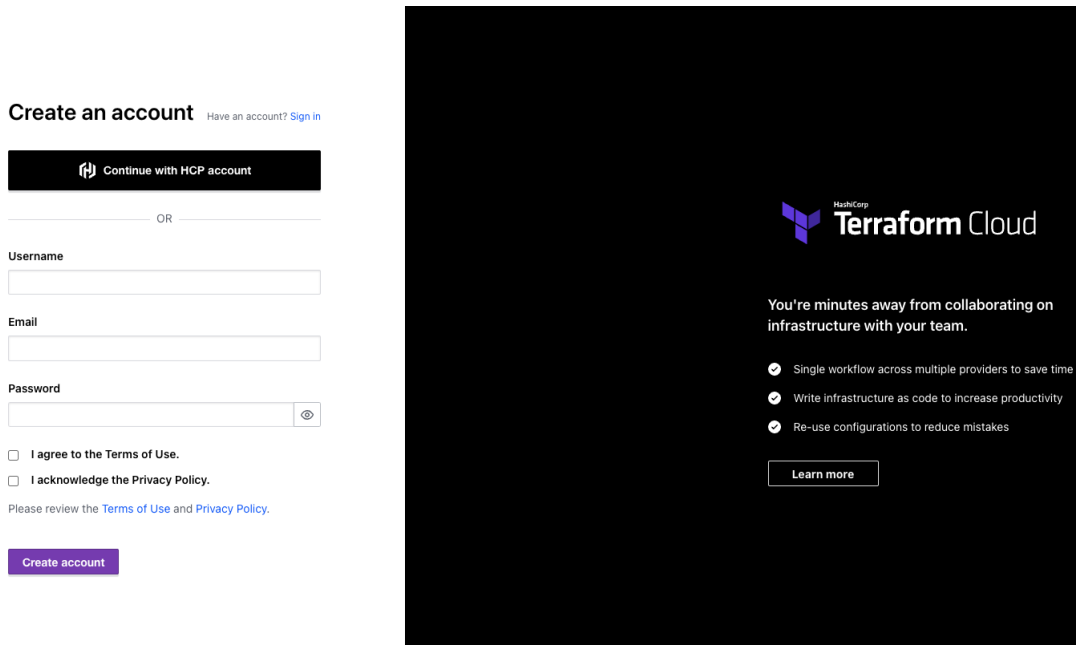


Imagen Nº 32: Formulario inicial de creación de cuenta.

Procedimiento:

1. Se debe completar todos los campos requeridos en el formulario de alta.
2. Una vez enviado el formulario de alta, es el momento de confirmar nuestra dirección de correo electrónico. En nuestra bandeja de entrada, encontraremos un mail que nos dará el enlace para la confirmación
3. Ingresamos los datos de la organización y nuestro email
4. Configuración inicial completa

Nota: Debido a la restricción extensión de la memoria, no es posible incluir, en la misma, guías de como instalar todas las aplicaciones, configuración o creación de cuentas. Sin embargo, todas las guías serán incluida en el directorio docs del repositorio eks-security-framework

6.1.2 Creación cuenta AWS Cloud

URL: https://aws.amazon.com/resources/create-account/

6.1.3 Creación cuenta de Kubescape

URL: https://cloud.armosec.io/account/sign-up

6.1.4 Generar un repositorios con la configuración de add-ons

Repositorio: eks-security-framework-add-ons

1. Generar un fork del repositorio base
2. Editar el fichero eks-blueprints-add-ons/chart/values.yaml
3. Para instalar el software se debe cambiar la propiedad enable a “true”

Notas:

- Todo el software será instalado por ArgoCD.
- Adicionalmente se puede agregar parámetros de configuración que existan en el helm chart
- La lista completa de aplicaciones soportadas , puede encontrarse en el directorio add-ons
- Es simple agregar nuevas aplicaciones, no existentes, se debe utilizar alguna ya existente como ejemplo.
- El repositorio original https://github.com/aws-samples/eks-blueprints-add-ons , es actualizado periódicamente con nuevas aplicaciones.

6.1.5 Generar un repositorios con la configuración base

Repositorio: eks-security-framework-base

1. Fork del repositorio

6.2.1 Preparación del repo

1. Hacer un fork o copia del repositorio “eks-security-terraform-workspace”
2. Agregar el siguiente código para la creación de la infraestructura (actualizar la configuración con nuestras preferencias)

```
module "cluster_infra" {
source = "app.terraform.io/security-framework/eks-security-clusters/aws"
version = "0.0.7"
```

```
cluster_name = "test-cluster"
vpc_cidr = "10.0.0.0/16"
repo_addons_url = "https://github.com/abelnieva/eks-blueprints-add-ons.git"
cluster_endpoint_public_access = true
tfc_agent_token = var.tfc_agent_token
ecr_repos_list = ["testrepo"]
dev_teams = {
dev_1_team = {
users = ["arn:aws:iam::481020473208:user/terraform"] #arns
namespaces = {
default = {
# Provides access to an existing namespace
create = false
}
}
}
}
kubescape_account_id = var.kubescape_account_id
repo_apps_url_dev = "git@github.com:abelnieva/eks-security-framework-apps.git"
repo_apps_pat_devh = "teams/dev_1_team/dev"
dns_zones = []
managed_node_groups = {
node_group_a = {
min_size = 1

max_size = 10
desired_size = 3
launch_template_os = "bottlerocket"
ami_type = "BOTTLEROCKET_x86_64"
instance_types = ["t3.large"]
capacity_type = "SPOT"

update_config = {
max_unavailable_percentage = 33 # or set `max_unavailable`
}

tags = {
ExtraTag = "example"
}
}
}
}
```

Variables

```
variable "kubescape_account_id" {
type = string
default = ""
description = "token to connect kubescape"

}
variable "tfc_agent_token" {
type = string
default = ""
description = "token to connect terraform cloud"

}
```

3. Crear workspace en terraform Cloud
    a. Abrir la plataforma https://app.terraform.io/
    b. Click en “Projects & workspaces”
    c. Click en “New”, Elegir Workspace
    d. Seleccionar “Version Control”
    e. Elegir un nombre al workspace
4. Configurar Granja de Agents para ejecutar nuestros workloads
    a. Ir a “Organization settings” => Agents(Security) => “Create Agent Pool”
    b. Designar un nombre a la granja
    c. Generar un token y guardarlo de manera segura.
5. Generar el número de cuenta en la pataforma ARMO https://www.armosec.io/
    a. Agregar cluster: Seleccionar “Settings” => “Clusters” => “Add Cluster”
    b. Copiar y guardar el valor de la variable “account=”
6. Crear claves de acceso programático en AWS para un usuario exclusivo para terraform. El usuario debe ser Administrador.
    a. Copiar y guardar los valores.
7. Configurar variables
    a. En los pasos anteriores hemos recolectado los valores que las variables deben tomar.
    b. Para el token de terraform cloud es “tfc_agent_token”
    c. Para el account id de ARMO es “kubescape_account_id”
    d. Claves de AWS se almacenan en “AWS_ACCESS_KEY_ID” y “AWS_SECRET_ACCESS_KEY”
8. Ejecución del Plan
    a. Seleccionar el workspace => Actions => Start new run => Start Run
9. Una vez ejecutado, el entorno nativo estará disponible listo para ser usado.
10. El código terraform es desplegado por terraform cloud, por otro lado el código compatible con manifiesto kubernetes, helm, kustomize etc, es realizado por ArgoCD.

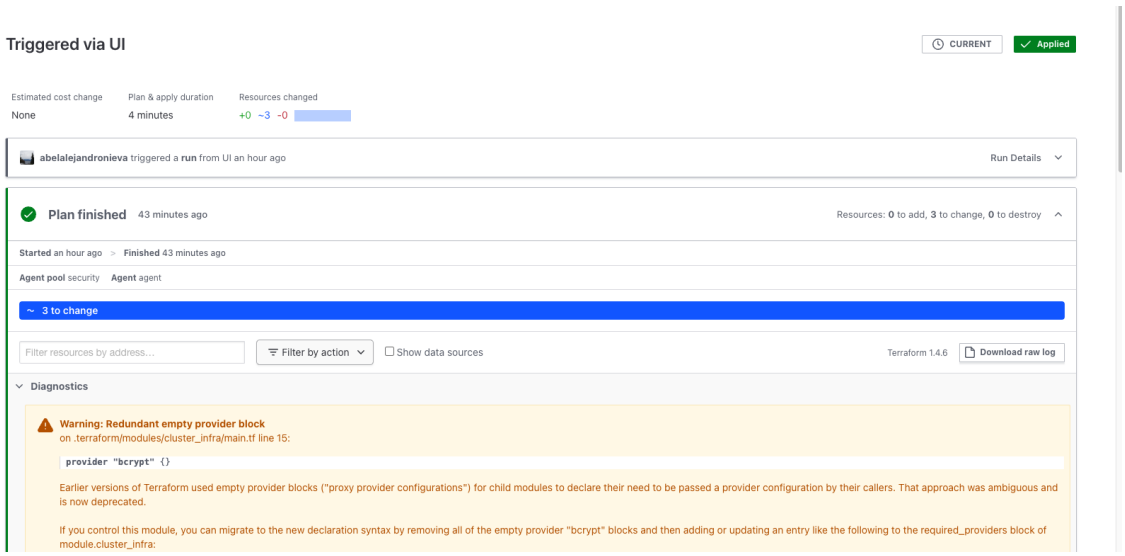


Imagen Nº 33: Pantalla de ejecución de planes.

6.2.2 ¿Cómo acceder a ArgoCD?

Para acceder a ArgoCD, podemos utilizar Lens o simplemente kubectl.

```
kubectl port-forward service/argo-cd-argocd-server 8080:80 -n argocd
```

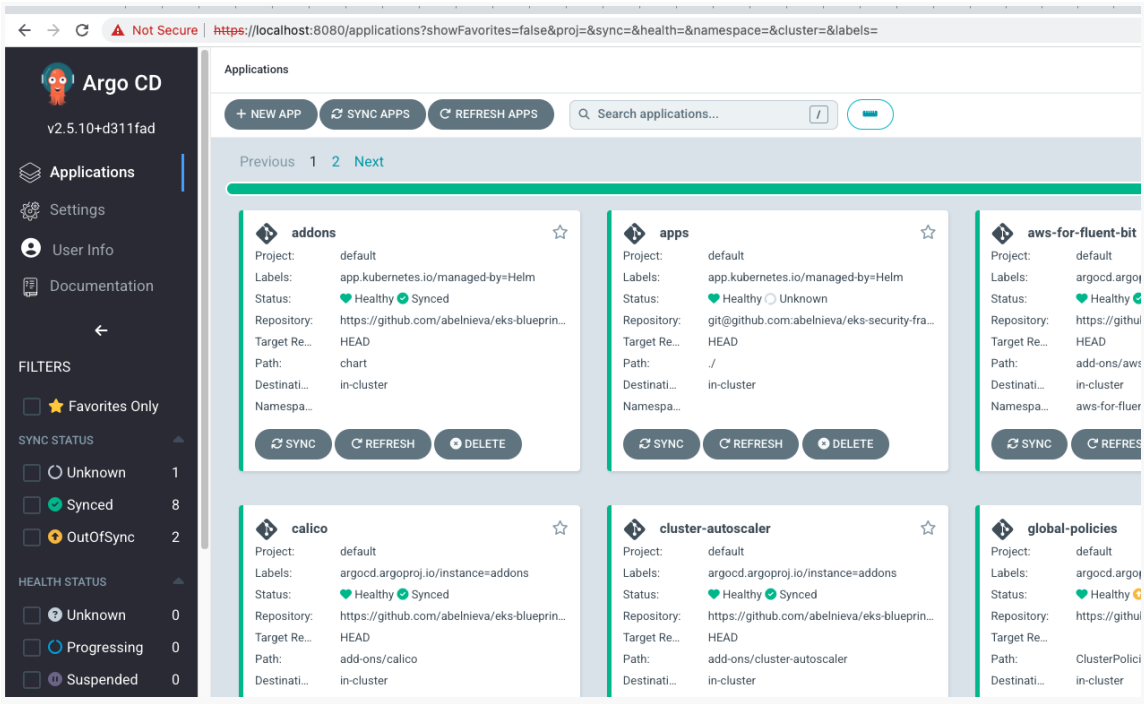


Imagen Nº 34: Pantalla de ArgoCD.

6.2.3 Despliegue aplicaciones

Repositorio: eks-security-framework-apps

El repositorio está divido por dos secciones

- envs
- teams

El fichero envs/dev/templates/dev_1_team.yaml, es una aplicación en formato ArgoCD. En el campo path se referencia, donde agregaremos los archivos de despliegue para ese equipo. Existe una separación entre entorno dev y prod.

Los archivos de dev y producción deberán ubicarse en ubicaciones diferentes, y referenciados por aplicaciones diferentes

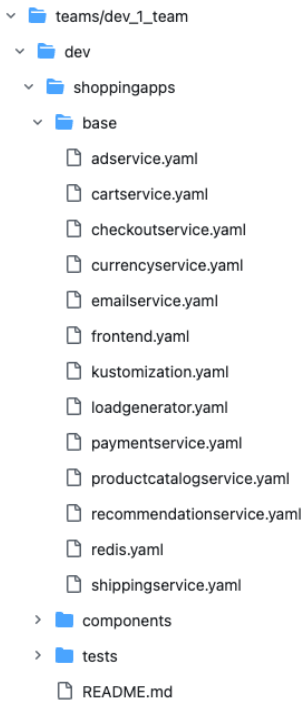


Imagen Nº 35: Distribución de ficheros de una aplicación

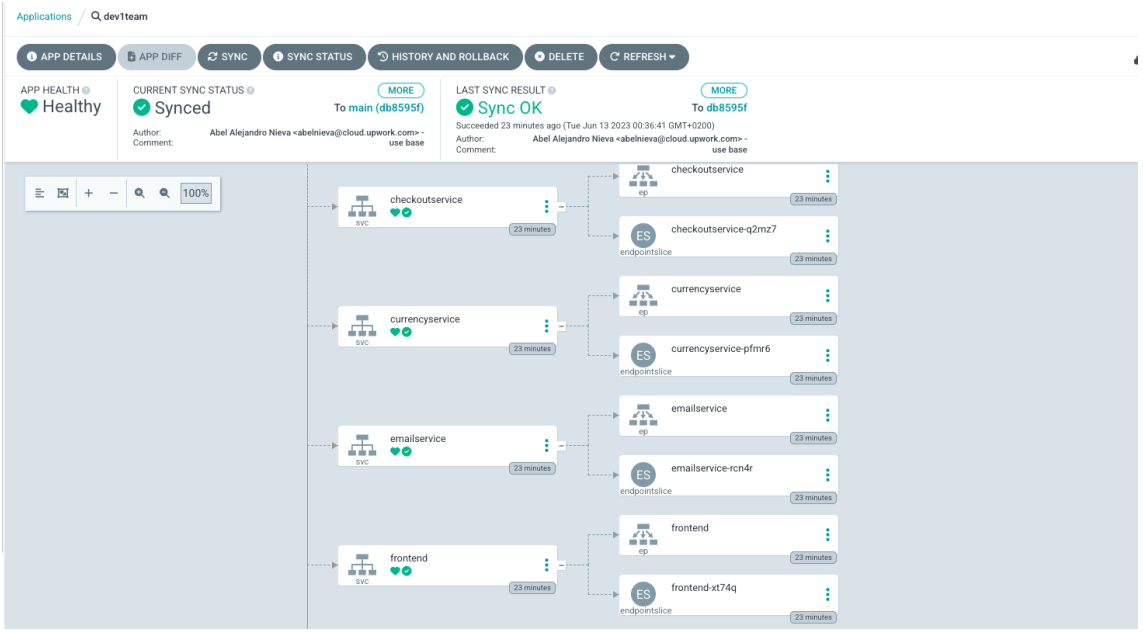


Imagen Nº 36: Aplicación desplegada por ArgoCD

6.2.4 Integración con Kubescape

Una vez desplegado nuestro entorno cloud, dispones de la integración con la plataforma ARMO , por medio de Kubescape

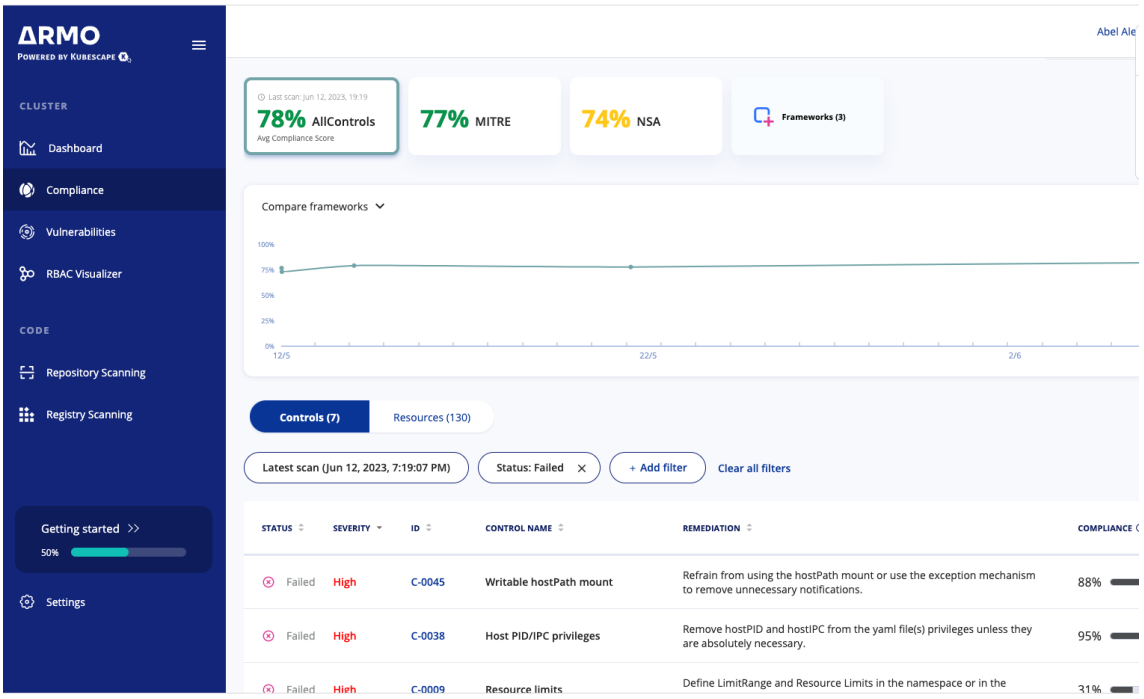


Imagen Nº 37: Plataforma ARMO

Con la información de la plataforma, podemos analizar las amenazas planteadas y resolverlas .

6.2.5 Integración en los pipelines

Por medio de GitHub Actions y la integración con terraform cloud es posible realizar verificaciones en las pull request

Existen tres verificaciones

- Pre-commit
- terrascan
- terraform cloud

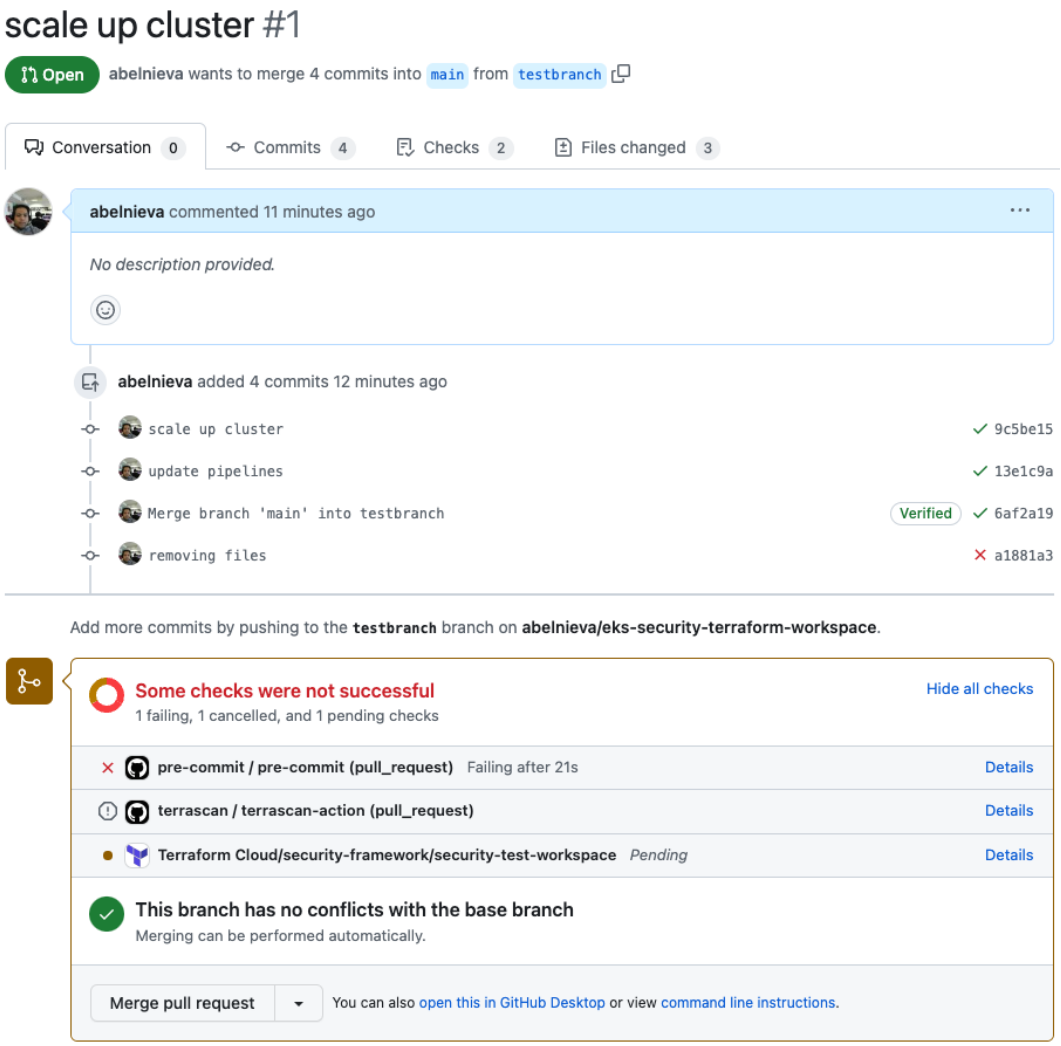


Imagen Nº 38: Pull request GitHub

6.2.6 Problemas conocidos

- Existe una limitación al crear un cluster con su API expuesta de manera privada. Es posible, pero la API debe ser pública durante la creación del cluster y la infraestructura. Posteriormente se puede configurar el workspace para que se ejecute en el agente, instalado dentro de nuestra infraestructura. La variable **cluster_endpoint_public_access** permite modificar la visibilidad del cluster.
- Debido al problema mencionado anteriormente, al momento de la destrucción del workspace, no se debe usar el agente de terraform cloud.
- DockerHub limita la descarga de imágenes, por lo tanto, es recomendable importar las imágenes a nuestro ECR, o configurar todas las aplicaciones a utilizar un secreto con un token de docker hub.

7. Conclusiones y trabajos futuros

Es notable la penetración de las aplicaciones nativas de la nube en la vida empresarial, las organizaciones día a día, desarrollan y mantienen la base de sus negocios con estas tecnologías. Kubernetes es definitivamente la solución que , día a día , suma más adeptos para el desarrollo de aplicaciones modernas. [54]A continuación se enumeran las conclusiones finales del trabajo de fin de máster.

1. Kubernetes y las aplicaciones nativas son la primera opción a la hora de desarrollar nuevos proyectos de software, pero su adopción en términos de seguridad genera grandes desafíos por parte de las organizaciones.
2. La inclusión de verificaciones a diferentes niveles, ej tanto antes del commit , como en el pipeline. Permiten tener más confianza en nuestros releases
3. Afortunadamente, las organizaciones, como los proveedores de servicios / software han comprendido que la seguridad en la nube es importante, y cada día nacen nuevos proyectos logran hacer un poco más simple la dura tarea de mantener la seguridad de nuestros entornos.
4. Existen muchos proyectos de código abierto y empresas brindan soporte comercial a estos proyectos, sin embargo, no hay una solución total para el manejo de la seguridad. Por lo tanto el uso de múltiples herramientas de diferentes fuentes es necesario para cubrir el abanico. Hay una imposibilidad de trabajar con proveedores únicos.
5. El uso de una metodología ágil para el desarrollo del producto ha sido de mucha utilidad, para poder llegar completar el desarrollo del framework.
6. Las empresas que brindan soluciones de pago basadas en proyectos de código abierto ofrecen soluciones muy superadoras, pero sus versiones gratuitas son muy limitadas.
7. La publicación como producto de código abierto del framework, generará impactos positivos a favor del cumplimiento de los objetivos de sostenibilidad, de la Agenda 2030. Me gustaría poner énfasis en el objetivo(ODS) 10 de “reducción de las desigualdades”, ya que organizaciones de todo el mundo tendrán acceso, y podrán beneficiarse de los conocimientos generados por este proyecto.

Debido a la naturaleza de k8s y las aplicaciones nativas son un área de conocimiento extensa, y en constante desarrollo, me gustaría plantear nuevas líneas investigativas, que podrían aprovechar las bases sentadas por el presente trabajo.

1. Estudio y desarrollo de políticas del controlador de admisión de k8s, como así también política de control en tiempo de ejecución en contenedores o nodos.
2. Estudio, desarrollo de despliegues de múltiples clusters en paralelo que funcionen de manera coordinada para brindar una alta disponibilidad de aplicaciones
3. Estudio y desarrollo de soluciones con un cluster, más cluster virtuales.
4. Llevar a cabo casos de estudios de implementación del framework en diversos ámbitos.

5. Estudio y desarrollo de una solución que permita facilitar la implementación de verificaciones en la cadena de suministro. Verificar el origen de imágenes por medio de la firma digital.

8. Glosario

Add-ons: Software de terceras partes a ser instalado
ArgoCD: Herramienta de código abierto para la implementación de GitOps en kubernetes
ARMO: Plataforma online para administrar información colectada por kubescape
API RESTful: Interfaz segura de comunicación entre servicios
AWS: Amazon web services, proveedor público de computación en la nube.
Calico: herramienta encargada de dotar a kubernetes la capacidad de comunicación segura (red) y establecimiento de políticas de red
CI/CD: Proceso de integración continua y entrega continua
CIDR: Estándar de red para interpretación de direcciones IP
CLI: Interfaz de línea de comando
Cluster: Conjunto de ordenadores que se gestionan juntos y participan en la gestión de carga de trabajo
Cloudwatch: Servicio incluido en Amazon Web Services que permite el almacenamiento de métricas de monitoreo y registros (logs) del cluster.
Commit: Acción fundamental en un controlador de versiones, como git, que permite incorporar nuevos cambios al repositorio
CNCF: Fundación que dirige proyectos de código abierto, cuyo objetivo es proporcionar herramientas y estándares a las organizaciones, para guiar la adopción de tecnologías nativas en la nube, para el desarrollo de aplicaciones modernas y escalables.
Containerd: Es un demonio que permite manejar contenedores en un nodo
Control Plane: Cerebro administrativo de un cluster kubernetes.
Dockershim: Interfaz de ejecución de contenedores en kubernetes para utilizar contenedores de tipo docker.
ECDHE/DHE: Protocolo criptográfico
ETCD: Almacén de datos más importante de Kubernetes, almacena y replica todos los estados de los recursos de Kubernetes en el clúster. Se trata de un elemento fundamental.
EKS: Amazon Elastic Kubernetes Service. Servicio administrado por Amazon, para el alojamiento de clusters kubernetes
Fargate: Fargate es una tecnología que provee a demanda, la capacidad computacional correcta para los contenedores. Con Fargate, no debemos preocuparnos por proveer, configurar o escalar grupos de máquinas virtuales.
Framework: Es un marco de trabajo que tiene como objetivo facilitar el desarrollo de infraestructura
Fork: Término usado por el mundo git para referirse a la acción de realizar una copia de un repositorio existente en uno nuevo conservando su contenido.
GitOps: Concepto derivado de la unión de las palabras GIT y OPS (operaciones), mantiene la infraestructura como código en un repositorio git y sólo despliega lo que fue declarado en git. Git es la fuente de la verdad.
Github actions: Plataforma de Integración y entrega continua de Github, que permite ejecutar flujos de trabajos en nuestros repositorios github
Hardening: El proceso de asegurar un sistema reduciendo su superficie de vulnerable

IAM: Administrador de identidades y accesos en AWS
Jenkins: Software popular para implementar pipelines de integración o despliegue continuo
K8s: Kubernetes, Porque existen 8 letras entre la K y la S
KubeArmor: Herramienta para la implementación de seguridad en cluster en tiempo de ejecución.
Kustomize: Herramienta del mundo de kubernetes que permite personalizar recursos
Lens: Entorno de administración de escritorio para clusters k8s
Linter: Herramienta, usualmente forma parte del análisis estático de código, que busca fallos de alineación o completitud del código. Esencial para determinar el nivel de completitud y cumplimiento que deseamos para el código almacenado en nuestros repositorios

ODS: Objetivos de desarrollo sustentable de la agenda 2030
OCI: Especificaciones generales sobre el desarrollo de contenedores
On premise: Referido a despliegues en infraestructura física propia del cliente, fuera de la nube.
Orquestadores: Sistema usado para automatizar despliegue, escalado y administración de aplicaciones en contenedores
Pipelines: Conjunto de pasos encadenados que permiten realizar de procesos de integración continua
RBAC: El control de acceso basado en roles (RBAC) es un mecanismo de control de acceso definido por roles y privilegios para determinar si a un usuario o cuenta de servicio se le debe otorgar acceso a un recurso.
Runc: Herramienta de línea de comando que permite la ejecución de contenedores con especificaciones OCI
Terraform: herramienta declaración y despliegue de infraestructura en la nube
TLS: Capa de seguridad de transporte de datos encriptado en las comunicaciones de red
Tfstate: Fichero de estado de Terraform
Visual studio code: Entorno de desarrollo creado por microsoft de gran popularidad
VPC endpoints: Concepto de red de AWS , que permite la creación de un punto de comunicación interno con la API de vpc .
Worker: Nodo de un cluster k8s donde se ejecutan las aplicaciones del usuario.
Zonas de disponibilidad (AZ): Ubicaciones, dentro de una región de, AWS separadas físicamente que en conjunto forman una región.

9. Bibliografía

[1] Wittkop, J. (2022). The Cybersecurity Playbook for Modern Enterprises.

[2] Red Hat. (2022). Understanding cloud-native applications. Recuperado de https://www.redhat.com/en/topics/cloud-native-apps

[3] VMWare. (s.f.). What is container orchestration? Recuperado de https://www.vmware.com/topics/glossary/content/container-orchestration.html

[4] Övergaard, G., & Palmkvist, K. (2004). Use Cases: Patterns and Blueprints.

[5] Jaramillo, D., Nguyen, D. V., & Smart, R. (2016). Leveraging microservices architecture by using Docker technology. En SoutheastCon 2016 (pp. 1-5). doi: 10.1109/SECON.2016.7506647.

[6] Poulton, N. (2020). Docker Deep Dive.

[7] Kocher, P. S. (2018). Microservices and Containers (1st ed.).

[8] Mitra, R., & Nadareishvili, I. (2020). Microservices: Up and Running.

[9] Trello. Recuperado de https://trello.com

[10] Hundhausen, R. (2021). Professional Scrum Development with Azure DevOps.

[11] Newman, S. (2021). Building microservices (2nd ed.).

[12] Wang, R. (2022). Infrastructure as Code, Patterns and Practices.

[13] Loeliger, J., & Ponuthorai, P. K. (2022). Version Control with Git (3rd ed.).

[14] Amazon Web Services. (2023). What is cloud computing? Recuperado de https://aws.amazon.com/what-is-cloud-computing/

[15] Lisdorf, A. (2021). Cloud Computing Basics: A Non-Technical Introduction.

[16] IBM. (2022). Costo de las filtraciones de datos. Recuperado de https://www.ibm.com/uk-en/security/data-breach

[17] Amazon Web Services. (2023). Shared Responsibility Model. Recuperado de https://aws.amazon.com/compliance/shared-responsibility-model/

[18] Amazon Web Services. (2023). Architecting HIPAA Security and Compliance on Amazon EKS. Recuperado de https://docs.aws.amazon.com/whitepapers/latest/architecting-hipaa-security-and-compliance-on-amazon-eks/aws-shared-responsibility-model.html

[19] Podjarny, G. (2021). Cloud Native Application Security.

[20] Cloud Native Computing Foundation (“CNCF”) Charter. Recuperado de https://github.com/cncf/foundation/blob/main/charter.md

[21] Podjarny, G. (2021). Cloud Native Application Security.

[22] Weaveworks. (2020). A practical guide to choosing between Docker containers and VMs. Recuperado de https://www.weave.works/blog/a-practical-guide-to-choosing-between-docker-containers-and-vms

[23] Poulton, N. (2000). Docker Deep Dive.

[24] Docker. (2023). Best practices for writing Dockerfiles. Recuperado de https://docs.docker.com/develop/develop-images/dockerfile_best-practices/

[25] Domingus, J., & Arundel, J. (2020). Cloud Native DevOps with Kubernetes (2nd ed.).

[26] Domingus, J., & Arundel, J. (2020). Cloud Native DevOps with Kubernetes (2nd ed.).

[27] Kubernetes. (2023). kube-controller-manager. Recuperado de https://kubernetes.io/docs/reference/command-line-tools-reference/kube-controller-manager/

[28] Kubernetes. (2023). Cloud Controller Manager. Recuperado de https://kubernetes.io/docs/concepts/architecture/cloud-controller/

[29] Kubernetes. (2023). Container Runtimes. Recuperado de https://kubernetes.io/docs/setup/production-environment/container-runtimes/

[30] Kubernetes. (2023). Pods. Recuperado de https://kubernetes.io/docs/concepts/workloads/pods/

[31] Densify. (s.f.). Kubernetes Node Capacity. Recuperado de https://www.densify.com/kubernetes-autoscaling/kubernetes-node-capacity

[32] ITPro Today. (s.f.). How to choose the right Kubernetes distribution. Recuperado de https://www.itprotoday.com/hybrid-cloud/how-choose-right-kubernetes-distribution

[33] Domingus, J., & Arundel, J. (2020). Cloud Native DevOps with Kubernetes (2nd ed.).

[34] Amazon Web Services. (s.f.). Amazon Elastic Kubernetes Service. Recuperado de https://docs.aws.amazon.com/whitepapers/latest/overview-deployment-options/amazon-elastic-kubernetes-service.html

[35] Brikman, Y. (2022). Terraform: Up and Running: Writing Infrastructure as Code.

[36] HashiCorp. (s.f.). Overview - Configuration language. Recuperado de https://developer.hashicorp.com/terraform/language

[37] HashiCorp. (s.f.). Terraform Registry. Recuperado de https://registry.terraform.io/

[38] HashiCorp. (s.f.). State. Recuperado de https://developer.hashicorp.com/terraform/language/state

[39] HashiCorp. (s.f.). Home - Terraform Cloud. Recuperado de https://developer.hashicorp.com/terraform/cloud-docs

[40] Armo. (s.f.). Kubescape Docs. Recuperado de https://hub.armosec.io/docs

[41] Tigera. (s.f.). About Calico. Recuperado de https://docs.tigera.io/calico/latest/about/about-calico

[42] Kyverno. (s.f.). Introduction. Recuperado de https://kyverno.io/docs/introduction/

[43] ArgoCD. (s.f.). Security. Recuperado de https://argo-cd.readthedocs.io/en/stable/operator-manual/security/

[44] Argo Project. (s.f.). Stronger Supply Chain Security Coming to Argo. Recuperado de https://blog.argoproj.io/security-supply-chain-security-57cffdb44967

[45] Cloudflare. (s.f.). ¿Qué es la "defensa en profundidad"? Recuperado de https://www.cloudflare.com/es-es/learning/security/glossary/what-is-defense-in-depth/

[46] Kubernetes. (s.f.). Overview of Cloud Native Security. Recuperado de https://kubernetes.io/docs/concepts/security/overview/

[47] Amazon Web Services. (2023). Infrastructure security in Amazon EKS. Recuperado de https://docs.aws.amazon.com/eks/latest/userguide/infrastructure-security.html

[48] Armo. (2023). How to avoid Kubernetes misconfigurations. Recuperado de https://www.armosec.io/blog/kubernetes-misconfigurations/

[49] Red Hat. (2023). A Guide to Secrets Management with GitOps and Kubernetes. Recuperado de https://cloud.redhat.com/blog/a-guide-to-secrets-management-with-gitops-and-kubernetes

[50] Amazon Web Services. (2023). AMI de Bottlerocket optimizadas para Amazon EKS. Recuperado de https://docs.aws.amazon.com/es_es/eks/latest/userguide/eks-optimized-ami-bottlerocket.html

[51] Red Hat. (2023). State of Kubernetes Security Report (Informe No. 262667-202304). Recuperado de https://www.redhat.com/rhdc/managed-files/cl-state-kubernetes-security-report-262667-202304-en.pdf

[52] OWASP Foundation. (s.f.). OWASP DevSecOps Guideline - v-0.2. Recuperado de https://owasp.org/www-project-devsecops-guideline/latest/01-Pre-commit

[53] Dawson, M., Burrell, D., Rahim, E., & Brewster, S. (2010). Integrating Software Assurance into the Software Development Life Cycle (SDLC). Journal of Information Systems Technology and Planning, 3, 49-53.

[54] Vaughan-Nichols, S. (2022). CNCF reports record Kubernetes and container adoption. Recuperado de https://www.zdnet.com/article/cncf-reports-record-kubernetes-and-container-adoption/

[55] Amazon Web Services. (s.f.). AWS Fargate. Recuperado de https://docs.aws.amazon.com/eks/latest/userguide/fargate.html

10. Acrónimos

AWS: Amazon Web Services
CNCF : Cloud Native Computing Foundation
EKS: Elastic Kubernetes Services
NSA : Agencia nacional de seguridad Estados unidos
OCI: Open container initiative
ODS: Objetivos de desarrollo sustentable
OWASP: Open Web Application Security Project

11. Anexos

Los repositorios resultantes de este trabajo pueden ser encontrados en github

terraform-aws-eks
url: https://github.com/terraform-aws-modules/terraform-aws-eks

terraform-aws-eks-security-clusters
url:https://github.com/abelnieva/terraform-aws-eks-security-clusters

eks-security-framework
url: https://github.com/abelnieva/eks-security-framework

eks-security-terraform-workspace
url: https://github.com/abelnieva/eks-security-terraform-workspace

eks-security-framework-base
url: https://github.com/abelnieva/eks-security-framework-base

eks-security-framework-apps
url : https://github.com/abelnieva/eks-security-framework-apps

eks-security-framework-add-ons
url: https://github.com/abelnieva/eks-security-framework-add-ons