Orquestación de servidores, Kurbenetes, Microservicios, OAuth2.0 e implementación de servicios en la nube (12-factor application)

HERRERA RODRIGUEZ JONATHAN BENJAMIN
SEMINARIO DE TECNOLOGÍAS DE INFORMACIÓN
7690-13-1131
jbherrerar9@miumg.edu.gt

1 Resumen

Explorar la orquestación en servidores me ha permitido comprender cómo se puede automatizar y gestionar de manera coordinada diversas tareas en entornos de servidores, lo que resulta en una operación más eficiente y menos propensa a errores. Kubernetes se destaca como una herramienta esencial en este proceso, especialmente cuando se trata de gestionar aplicaciones en contenedores. Esta plataforma no solo facilita la portabilidad entre diferentes infraestructuras, sino que también soporta la arquitectura basada en microservicios, que ofrece una flexibilidad y escalabilidad mucho mayores en comparación con los sistemas monolíticos tradicionales. Además, me llamó la atención cómo la seguridad entre microservicios se maneja con técnicas como mTLS, asegurando la comunicación segura entre ellos. Por otro lado, OAuth 2.0, un estándar de autorización, permite que las aplicaciones accedan a recursos en nombre de los usuarios sin comprometer sus credenciales, utilizando tokens de acceso para mantener un control riguroso y seguro en la interacción entre aplicaciones y recursos.

2 Palabras Clave

Tokens, Automatización, Contenedores, Microservicios, Portabilidad.

3 Desarrollo del Tema

Orquestación En Servidores

¿Qué es la orquestación en servidores?
La orquestación en servidores se refiere a la capacidad de automatizar y gestionar de manera coordinada una serie de tareas y procesos en un entorno de servidores. Esto implica la implementación y gestión de aplicaciones, servicios, configuraciones y recursos de forma eficiente y coherente.
¿Para qué sirve la orquestación en servidores?
La orquestación en servidores sirve para simplificar y agilizar las operaciones de TI, permitiendo a las organizaciones gestionar y escalar sus infraestructuras de manera eficiente. Algunos de los casos de uso comunes de la orquestación en servidores son:

- Implementación de aplicaciones: La orquestación permite la implementación automatizada de aplicaciones y servicios en entornos de servidores, garantizando una implementación rápida y consistente.

- Gestión de configuraciones: La orquestación facilita la gestión y configuración centralizada de los servidores y recursos asociados, asegurando una configuración coherente y reduciendo los errores manuales.
- Escalabilidad y administración de recursos: La orquestación permite escalar y administrar eficientemente los recursos de servidores, adaptándose a la demanda variable y optimizando el uso de recursos disponibles.
- Automatización de tareas: La orquestación automatiza tareas repetitivas y tediosas, liberando tiempo y recursos para actividades más estratégicas y de alto valor.

¿Cómo funciona la orquestación en servidores?
La orquestación en servidores se basa en herramientas y plataformas de gestión que permiten definir y ejecutar flujos de trabajo automatizados. Estas herramientas proporcionan capacidades para la creación de scripts, la gestión de configuraciones, la programación y la supervisión de tareas.
La orquestación se realiza a través de un controlador centralizado que coordina y ejecuta los flujos de trabajo definidos. El controlador se comunica con los servidores y recursos involucrados para implementar y gestionar las tareas de forma coherente y automatizada.
Beneficios de la orquestación en servidores

- Eficiencia operativa: La orquestación automatiza tareas y procesos, lo que reduce la necesidad de intervención manual y minimiza los errores humanos, mejorando la eficiencia y reduciendo el tiempo de inactividad.
- Escalabilidad y flexibilidad: La orquestación permite escalar y gestionar los recursos de manera dinámica, adaptándose a las demandas cambiantes y garantizando un rendimiento óptimo.
- Consistencia y conformidad: La orquestación asegura que los servidores y recursos se configuran y gestionan de manera coherente, cumpliendo con las políticas y estándares establecidos.
- Agilidad y rápida implementación: La orquestación permite una implementación rápida y consistente de aplicaciones y servicios, acelerando el tiempo de comercialización y la capacidad de respuesta a las necesidades del negocio.

[4], [3]

Kubernetes

¿Qué es Kubernetes? Kubernetes es una plataforma portable y extensible de código abierto para administrar cargas de trabajo y servicios. Kubernetes facilita la automatización y la configuración declarativa. Tiene un ecosistema grande y en rápido crecimiento. Google liberó el proyecto Kubernetes en el año 2014. Kubernetes se basa en la experiencia de Google corriendo aplicaciones en producción a gran escala por década y media, junto a las mejores ideas y prácticas de la comunidad. ¿Por qué necesito Kubernetes y qué puede hacer por mi? Kubernetes tiene varias características. Puedes pensar en Kubernetes como:

- Una plataforma de contenedores
- Una plataforma de microservicios

- Una plataforma portable de nube

y mucho más. Kubernetes ofrece un entorno de administración centrado en contenedores. Kubernetes orquesta la infraestructura de cómputo, redes y almacenamiento para que las cargas de trabajo de los usuarios no tengan que hacerlo. Esto ofrece la simplicidad de las Plataformas como Servicio (PaaS) con la flexibilidad de la Infraestructura como Servicio (IaaS) y permite la portabilidad entre proveedores de infraestructura. ¿Qué hace de Kubernetes una plataforma? A pesar de que Kubernetes ya ofrece muchas funcionalidades, siempre hay nuevos escenarios que se benefician de nuevas características. Los flujos de trabajo de las aplicaciones pueden optimizarse para acelerar el tiempo de desarrollo. Una solución de orquestación propia puede ser suficiente al principio, pero suele requerir una automatización robusta cuando necesita escalar. Además, el Plano de Control de Kubernetes usa las mismas APIs que usan los desarrolladores y usuarios finales. Los usuarios pueden escribir sus propios controladores, como por ejemplo un planificador o scheduler, usando sus propias APIs desde una herramienta de línea de comandos. Este diseño ha permitido que otros sistemas sean construidos sobre Kubernetes.
Lo que Kubernetes no es: Kubernetes no es una Plataforma como Servicio (PaaS) convencional. Ya que Kubernetes opera a nivel del contenedor y no a nivel del hardware, ofrece algunas características que las PaaS también ofrecen, como deployments, escalado, balanceo de carga, registros y monitoreo. Dicho esto, Kubernetes no es monolítico y las soluciones que se ofrecen de forma predeterminada son opcionales e intercambiables. Kubernetes ofrece los elementos esenciales para construir una plataforma para desarrolladores, preservando la elección del usuario y la flexibilidad en las partes más importantes. Entonces, podemos decir que Kubernetes:

- No limita el tipo de aplicaciones que soporta. Kubernetes busca dar soporte a un número diverso de cargas de trabajo, que incluyen aplicaciones con y sin estado así como aplicaciones que procesan datos. Si la aplicación puede correr en un contenedor, debería correr bien en Kubernetes.

- No hace deployment de código fuente ni compila tu aplicación. Los flujos de integración, entrega y deployment continuo (CI/CD) vienen determinados por la cultura y preferencia organizacional y sus requerimientos técnicos.

- No provee servicios en capa de aplicación como middleware (por ejemplo, buses de mensaje), frameworks de procesamiento de datos (como Spark), bases de datos (como MySQL), caches o sistemas de almacenamiento (como Ceph). Es posible correr estas aplicaciones en Kubernetes, o acceder a ellos desde una aplicación usando un mecanismo portable como el Open Service Broker.

- No dictamina las soluciones de registros, monitoreo o alerta que se deben usar. Hay algunas integraciones que se ofrecen como prueba de concepto, y existen mecanismos para recolectar y exportar métricas.

- No provee ni obliga a usar un sistema o lenguaje de configuración (como jsonnet) sino que ofrece una API declarativa que puede ser usada con cualquier forma de especificación declarativa.

- No provee ni adopta un sistema exhaustivo de mantenimiento, administración o corrección automática de errores.

¿Por qué usar contenedores? ¿Te preguntas las razones para usar contenedores? La Manera Antigua de desplegar aplicaciones era instalarlas en un servidor usando el administrador de paquetes

del sistema operativo. La desventaja era que los ejecutables, la configuración, las librerías y el ciclo de vida de todos estos componentes se entretejían unos a otros. Podíamos construir imágenes de máquina virtual inmutables para tener rollouts y rollbacks predecibles, pero las máquinas virtuales son pesadas y poco portables. La Manera Nueva es desplegar contenedores basados en virtualización a nivel del sistema operativo, en vez del hardware. Estos contenedores están aislados entre ellos y con el servidor anfitrión: tienen sus propios sistemas de archivos, no ven los procesos de los demás y el uso de recursos puede ser limitado. Son más fáciles de construir que una máquina virtual, y porque no están acoplados a la infraestructura y sistema de archivos del anfitrión, pueden llevarse entre nubes y distribuciones de sistema operativo. En resumen, los beneficios de usar contenedores incluyen:

- Ágil creación y despliegue de aplicaciones: Mayor facilidad y eficiencia al crear imágenes de contenedor en vez de máquinas virtuales
- Desarrollo, integración y despliegue continuo: Permite que la imagen de contenedor se construya y despliegue de forma frecuente y confiable, facilitando los rollbacks pues la imagen es inmutable
- Separación de tareas entre Dev y Ops: Puedes crear imágenes de contenedor al momento de compilar y no al desplegar, desacoplando la aplicación de la infraestructura
- Observabilidad No solamente se presenta la información y métricas del sistema operativo, sino la salud de la aplicación y otras señales
- Consistencia entre los entornos de desarrollo, pruebas y producción: La aplicación funciona igual en un laptop y en la nube
- Portabilidad entre nubes y distribuciones: Funciona en Ubuntu, RHEL, CoreOS, tu datacenter físico, Google Kubernetes Engine y todo lo demás
- Administración centrada en la aplicación: Eleva el nivel de abstracción del sistema operativo y el hardware virtualizado a la aplicación que funciona en un sistema con recursos lógicos
- Microservicios distribuidos, elásticos, liberados y débilmente acoplados: Las aplicaciones se separan en piezas pequeñas e independientes que pueden ser desplegadas y administradas de forma dinámica, y no como una aplicación monolítica que opera en una sola máquina de gran capacidad
- Aislamiento de recursos: Hace el rendimiento de la aplicación más predecible
- Utilización de recursos: Permite mayor eficiencia y densidad

[6]

Microservicios

Son módulos ligeros con acople suelto que pueden servir como componentes básicos de las aplicaciones complejas basadas en la nube. Aunque los microservicios individuales pueden operar de forma independiente, tienen un acople suelto en una interfaz unificada. La arquitectura de microservicios se considera un reemplazo moderno y flexible del modelo de desarrollo más tradicional de la arquitectura monolítica. En una arquitectura monolítica, generalmente hay un servidor virtualizado o físico asignado a cada aplicación y esos servidores siempre están en ejecución.

La disponibilidad y el escalamiento de la aplicación dependen completamente del hardware subyacente, que es una entidad fija. Los microservicios, por el contrario, pueden operar varias instancias en un solo servidor o en varios servidores, a medida que los recursos escalan de forma dinámica para admitir las demandas de la carga de trabajo. Los microservicios individuales suelen estar en contenedores para mejorar la portabilidad y la escalabilidad.
Las diferentes características de los microservicios Como proceso de desarrollo, el marco de microservicios tiene ciertas características que son comunes, pero no universales. Esas características incluyen:

- Componentes. Los microservicios suelen estar formados por componentes de software discretos que son actualizables y reemplazables de forma individual.
- Servicios. Los componentes comprenden servicios que están disponibles para comunicarse a pedido, pero pueden no estar activos de forma continua entre las solicitudes o las llamadas.
- Implementación independiente. En su mayor parte, los componentes de servicios individuales operan de forma independiente uno de otro dentro del marco de microservicios.
- Seguridad. La comunicación entre los microservicios suele estar cifrada con seguridad de la capa de transporte mutuo (mTLS) para proteger los datos del malware y las intrusiones mientras está en tránsito.
- Contenerización. Los microservicios suelen implementarse en contenedores para lograr escalabilidad y portabilidad adicionales.

Beneficios de los microservicios y las aplicaciones de la nube Los microservicios continúan ganando popularidad debido a la flexibilidad innata de la arquitectura en la nube. De hecho, un estudio reciente realizado por Intel reveló que el 83 Los microservicios también plantean algunos desafíos de implementación. Los módulos individuales pueden ejecutarse en diferentes sistemas para cumplir una solicitud, lo que puede producir que el desempeño sea inconsistente entre los módulos. Del mismo modo, la latencia puede convertirse en un problema para algunos microservicios, como cuando hay picos de demanda. Estos problemas a menudo pueden resolverse mediante el sobreaprovisionamiento de recursos para acaparar los niveles de demanda pico. Cómo funciona la arquitectura de microservicios En una arquitectura de microservicios, una aplicación compleja se desglosa en capacidades discretas que pueden desarrollarse y operarse de forma independiente. Los microservicios individuales se comunican entre sí, a menudo a través de las APIs, y tienen conexiones sueltas, pero cada microservicio puede desarrollarse, implementarse y actualizarse por separado. La implementación de microservicios suele seguir uno de tres patrones:

1. Nativo de la nube. Algunos servicios y aplicaciones establecidos de alto volumen comienzan como microservicios y permanecen en la nube. Según International Data Corporation (IDC), alrededor del 56% de los microservicios son nativos de la nube, mientras que el 44% restante se originaron como aplicaciones antiguas.
2. Refactorización y cambio. Estas implementaciones comienzan en las instalaciones o en un centro de datos en el perímetro y se refactorizan para adaptarse a la arquitectura de microservicios basados en la nube. La refactorización puede incluir reasignar bases de datos y otros recursos asociados con la arquitectura monolítica, de modo que se combinen con los microservicios correspondientes.

3. Lift and shift. Algunas organizaciones migran sus aplicaciones a una arquitectura de microservicios, sin refactorización, en una simple transición “lift and shift”.

Protección de los microservicios Debido a que los microservicios suelen estar diseñados para interactuar entre sí, es importante proteger los datos mientras se usan en cada extremo de la interacción o en tránsito entre esos dos puntos. El cifrado con la seguridad de la capa de transporte mutuo (mTLS) es una solución común que ayuda a reducir el riesgo de intrusión y malware para los datos en cada punto de conexión y en tránsito. El objetivo de la mTLS es cifrar todas las solicitudes que haga cada microservicio. Este enfoque holístico para la seguridad es la base de un entorno de confianza cero, donde cada microservicio, usuario y conexión debe verificarse y autorizarse de forma independiente. Cuándo usar los microservicios Un enfoque de microservicios puede ayudar a migrar y escalar las aplicaciones mediante el desglose de las soluciones más complejas en sus partes integrantes. Cada microservicio se desarrolla e implementa de forma independiente, y los diversos microservicios operan juntos como una entidad integrada de forma suelta. Las organizaciones tienen más probabilidades de usar los microservicios cuando desarrollan una nueva solución nativa de nube. También pueden implementar microservicios cuando una arquitectura monolítica existente se vuelve muy poco manejable para satisfacer las necesidades cambiantes. La transición a una arquitectura de microservicios puede producir cambios en la organización y la estructura de los equipos mismos de DevOps Con el tiempo, los equipos pueden alinearse con su nuevo enfoque interfuncional en las capacidades empresariales en lugar de los silos que imitan las capas funcionales en la pila de tecnología. [5]

OAuth 2.0

Qué es OAuth 2.0? OAuth 2.0, que significa “Open Authorization” (autorización abierta), es un estándar diseñado para permitir que un sitio web o una aplicación accedan a recursos alojados por otras aplicaciones web en nombre de un usuario. Sustituyó a OAuth 1.0 en 2012 y ahora es el estándar de facto de la industria para la autorización en línea. OAuth 2.0 proporciona acceso consentido y restringe las acciones que la aplicación del cliente puede realizar en los recursos en nombre del usuario, sin compartir nunca las credenciales del usuario. Principios de OAuth2.0 OAuth 2.0 es un protocolo de autorización y NO un protocolo de autenticación. Como tal, está diseñado principalmente como un medio para conceder acceso a un conjunto de recursos, por ejemplo, API remotas o datos de usuario. Auth 2.0 utiliza tokens de acceso. Un Token de acceso es un dato que representa la autorización para acceder a los recursos en nombre del usuario final. OAuth 2.0 no define un formato específico para los tokens de acceso. Sin embargo, en algunos contextos, se suele utilizar el formato JSON Web Token (JWT). Esto permite a los emisores de tokens incluir datos en el propio token. Además, por razones de seguridad, los tokens de acceso pueden tener una fecha de caducidad. Roles de OAuth2.0 La idea de los roles forma parte de la especificación central del marco de autorización OAuth2.0. Estos definen los componentes esenciales de un sistema de OAuth 2.0, y son los siguientes:

- Propietario del recurso: El usuario o sistema que posee los recursos protegidos y puede conceder acceso a ellos.
- Cliente: El cliente es el sistema que requiere acceso a los recursos protegidos. Para acceder a los recursos, el cliente debe poseer el token de acceso correspondiente.
- Servidor de autorización: Este servidor recibe las solicitudes de tokens de acceso del cliente y las emite una vez que el propietario del recurso se ha autenticado y ha dado su consentimiento. El servidor de autorización expone dos puntos de conexión: el punto de conexión

de autorización, que maneja la autenticación interactiva y el consentimiento del usuario, y el punto de conexión de token, que está involucrado en una interacción de máquina a máquina.

- Servidor de recursos: Un servidor que protege los recursos del usuario y recibe las solicitudes de acceso del cliente. Acepta y valida un token de acceso del cliente y le devuelve los recursos adecuados.

Ámbitos de OAuth 2.0 Los ámbitos son un concepto importante en OAuth 2.0. Se utilizan para especificar exactamente el motivo por el que se puede conceder el acceso a los recursos. Los valores del ámbito aceptables, y los recursos a los que se refieren, dependen del servidor de recursos. Tokens de acceso y código de autorización de OAuth 2.0 El servidor de autorización de OAuth 2 no puede devolver directamente un token de acceso después de que el propietario del recurso haya autorizado el acceso. En su lugar, y para mayor seguridad, se puede devolver un Código de Autorización, que se cambia por un token de acceso. Además, el servidor de autorización también puede emitir un Token de actualización con el token de acceso. ¿Cómo funciona OAuth 2.0? En el nivel más básico, antes de poder utilizar OAuth 2.0, el cliente debe adquirir sus propias credenciales, un id de cliente y un client secret, del servidor de autorización para identificarse y autenticarse al solicitar un token de acceso. La solicitud, el intercambio y la respuesta de los tokens siguen el siguiente flujo general:

1. El cliente solicita autorización (solicitud de autorización) al servidor de autorización, proporcionando el id y el client secret como identificación; también proporciona los ámbitos y un URI de extremo (URI de redireccionamiento) al que enviar el token de acceso o el código de autorización.

2. El servidor de autorización autentica al cliente y verifica que los ámbitos solicitados están permitidos.

3. El propietario del recurso interactúa con el servidor de autorización para conceder el acceso.

4. El servidor de autorización redirige de vuelta al cliente con un código de autorización o un token de acceso, según el tipo de concesión, como se explicará en la siguiente sección. También puede devolverse un token de actualización.

5. Con el token de acceso, el cliente solicita acceso al recurso desde el servidor de recursos.

Tipos de concesión en OAuth 2.0 En OAuth 2.0, las concesiones son el conjunto de pasos que un cliente tiene que realizar para obtener la autorización de acceso a los recursos. El marco de autorización proporciona varios tipos de concesión para hacer frente a diferentes escenarios:

- Concesión del Código de autorización: El servidor de autorización devuelve al cliente un código de autorización de un solo uso, que se intercambia por un token de acceso. Esta es la mejor opción para las aplicaciones web tradicionales en las que el intercambio puede realizarse de forma segura en el lado del servidor.

- Concesión Implícita: Un flujo simplificado en el que el token de acceso se devuelve directamente al cliente. En el flujo implícito, el servidor de autorización puede devolver el token de acceso como un parámetro en el URI de devolución de llamada o como una respuesta a la publicación de un formulario. La primera opción está ahora obsoleta debido a la posible fuga de tokens.

- Concesión del código de autorización con clave de prueba para el intercambio de códigos (PKCE): Este flujo de autorización es similar a la concesión del Código de autorización, pero con pasos adicionales que lo hacen más seguro para las aplicaciones móviles/nativas y SPA.
- Tipo de concesión de credenciales del propietario del recurso: Esta concesión requiere que el cliente adquiera primero las credenciales del propietario del recurso, que se pasan al servidor de autorización.
- Tipo de concesión de credenciales de clientes: Se utiliza para aplicaciones no interactivas, por ejemplo, procesos automatizados, microservicios, etc. En este caso, la aplicación se autentica por sí misma mediante el uso de su id de cliente y su client secret.
- Flujo de autorización de dispositivos: Una concesión que permite el uso de aplicaciones en dispositivos con restricciones de entrada, como las televisiones inteligentes.
- Concesión del token de actualización: El flujo que implica el intercambio de un token de actualización por un nuevo token de acceso.

[2]

Implementación de servicios en la nube (12-factor application)

El software se está distribuyendo como un servicio: se le denomina web apps, o software as a service (SaaS). "The twelve-factor app" es una metodología para construir aplicaciones SaaS que:

- Usan formatos declarativos para la automatización de la configuración, para minimizar el tiempo y el coste que supone que nuevos desarrolladores se unan al proyecto;
- Tienen un contrato claro con el sistema operativo sobre el que trabajan, ofreciendo la máxima portabilidad entre los diferentes entornos de ejecución;
- Son apropiadas para desplegarse en modernas plataformas en la nube, obviando la necesidad de servidores y administración de sistemas;
- Minimizan las diferencias entre los entornos de desarrollo y producción, posibilitando un despliegue continuo para conseguir la máxima agilidad;
- Y pueden escalar sin cambios significativos para las herramientas, la arquitectura o las prácticas de desarrollo.

La metodología "twelve-factor" puede ser aplicada a aplicaciones escritas en cualquier lenguaje de programación, y cualquier combinación de 'backing services' (bases de datos, colas, memoria cache, etc). Contexto Los colaboradores de este documento han estado involucrados directamente en el desarrollo y despliegue de cientos de aplicaciones, y han sido testigos indirectos del desarrollo, las operaciones y el escalado de cientos de miles de aplicaciones mediante nuestro trabajo en la plataforma Heroku. Este documento sintetiza toda nuestra experiencia y observaciones en una amplia variedad de aplicaciones SaaS. Es la triangulación entre practicas ideales para el desarrollo de aplicaciones, prestando especial atención a las dinámicas del crecimiento natural de una aplicación a lo largo del tiempo, las dinámicas de colaboración entre desarrolladores que trabajan en el código base de las aplicaciones y evitando el coste de la entropía del software. Nuestra motivación es mejorar la concienciación sobre algunos problemas sistémicos que hemos observado en el desarrollo de las aplicaciones modernas, aportar un vocabulario común que sirva para

discutir sobre estos problemas, y ofrecer un conjunto de soluciones conceptualmente robustas para esos problemas acompañados de su correspondiente terminología. El formato está inspirado en los libros de Martin Fowler Patterns of Enterprise Application Architecture y Refactoring.
Twelve Factors

1. I. Código base (Codebase) Un código base sobre el que hacer el control de versiones y multiples despliegues
2. II. Dependencias Declarar y aislar explícitamente las dependencias
3. III. Configuraciones Guardar la configuración en el entorno
4. IV. Backing services Tratar a los "backing services" como recursos conectables
5. V. Construir, desplegar, ejecutar Separar completamente la etapa de construcción de la etapa de ejecución
6. VI. Procesos Ejecutar la aplicación como uno o más procesos sin estado
7. VII. Asignación de puertos Publicar servicios mediante asignación de puertos
8. VIII. Concurrencia Escalar mediante el modelo de procesos
9. IX. Desechabilidad Hacer el sistema más robusto intentando conseguir inicios rápidos y finalizaciones seguras
10. X. Paridad en desarrollo y producción Mantener desarrollo, preproducción y producción tan parecidos como sea posible
11. XI. Historiales Tratar los historiales como una transmisión de eventos
12. XII. Administración de procesos Ejecutar las tareas de gestión/administración como procesos que solo se ejecutan una vez

[1]

Observación

1. Kubernetes es la solución más reconocida para la orquestación de contenedores, es interesante ver cómo existen alternativas con enfoques específicos para diferentes necesidades, lo que destaca la importancia de elegir la herramienta adecuada según el caso de uso.
2. La implementación de seguridad en la orquestación de contenedores, especialmente con tecnologías como mTLS y OAuth 2.0, requiere un conocimiento profundo y una configuración meticulosa, lo que puede ser un desafío para equipos con menos experiencia en estas áreas.
3. La capacidad de los contenedores para garantizar la portabilidad de aplicaciones a través de diferentes entornos es un factor clave en su adopción, especialmente en un mundo donde la flexibilidad y la escalabilidad son esenciales.

Comentario

1. En el enfoque de la automatizacion me llamó la atención cómo la automatización se convierte en un pilar central en la orquestación de contenedores, permitiendo que los equipos se enfoquen en tareas de mayor valor en lugar de preocuparse por la gestión manual de infraestructuras complejas.

2. La seguridad sigue siendo una preocupación principal en la implementación de arquitecturas de microservicios, y la integración de protocolos como mTLS y OAuth 2.0 no solo es necesaria, sino que debe ser una prioridad desde el inicio del diseño.

Conclusiones

1. La orquestación y la automatización integrar la orquestación de contenedores en un entorno de desarrollo no solo mejora la eficiencia operativa, sino que también permite una mejor gestión de los recursos, haciendo que la automatización sea una necesidad más que una opción.

2. La seguridad y la funcionalidad, nos dejan un enfoque a estar progtegidos hablando un poco de seguridad es crítica, debe existir un equilibrio entre implementar medidas robustas y mantener la funcionalidad y agilidad en el desarrollo. Encontrar este balance es clave para el éxito de cualquier proyecto en la nube.

3. La rápida evolución y adopción de nuevas tecnologías en la orquestación de contenedores destaca la necesidad de mantenerse actualizado con las mejores prácticas y herramientas emergentes para aprovechar al máximo sus beneficios.

4. La capacidad de escalar aplicaciones de manera eficiente y confiable es uno de los mayores beneficios de los contenedores. A medida que las organizaciones crecen, esta característica se vuelve fundamental para soportar la demanda y mantener un rendimiento óptimo.

References

[1] 12Factor. Los 12 factores. Accedido: Agosto 2024.

[2] Auth0. ¿qué es oauth 2? Accedido: Agosto 2024.

[3] Huawei Forum. Orquestación en servidores. Accedido: Agosto 2024.

[4] IBM. ¿qué es la orquestación de contenedores?, s/f. Recuperado el 21 de junio de 2023.

[5] Intel. Microservicios. Accedido: Agosto 2024.

[6] Kubernetes. ¿qué es kubernetes? Accedido: Agosto 2024.

Repositorio Git

- `https://github.com/JohnH31/SEMINARIO-DE-TECNOLOG-AS-DE-INFORMACI-N.git`