Comprensión e implementación de microinterfaces en AWS

AWS Guía prescriptiva

AWS Guía prescriptiva: Comprensión e implementación de microinterfaces en AWS

Table of Contents

Comprensión e implementación de microinterfaces en AWS

Amazon Web Services (colaboradores)

Julio de 2024 (historial de documentos)

A medida que las organizaciones se esfuerzan por lograr agilidad y escalabilidad, la arquitectura monolítica convencional a menudo se convierte en un obstáculo, lo que dificulta el desarrollo y la implementación rápidos. Las microinterfaces mitigan esta situación al dividir las interfaces de usuario complejas en componentes más pequeños e independientes que se pueden desarrollar, probar e implementar de forma autónoma. Este enfoque mejora la eficiencia de los equipos de desarrollo y facilita la colaboración entre el backend y el frontend, lo que fomenta la alineación de los sistemas distribuidos. end-to-end

Esta guía prescriptiva está diseñada para ayudar a los líderes de TI, propietarios de productos y arquitectos de diversos ámbitos profesionales a comprender la arquitectura microfrontend y crear aplicaciones microfrontend en Amazon Web Services ().AWS

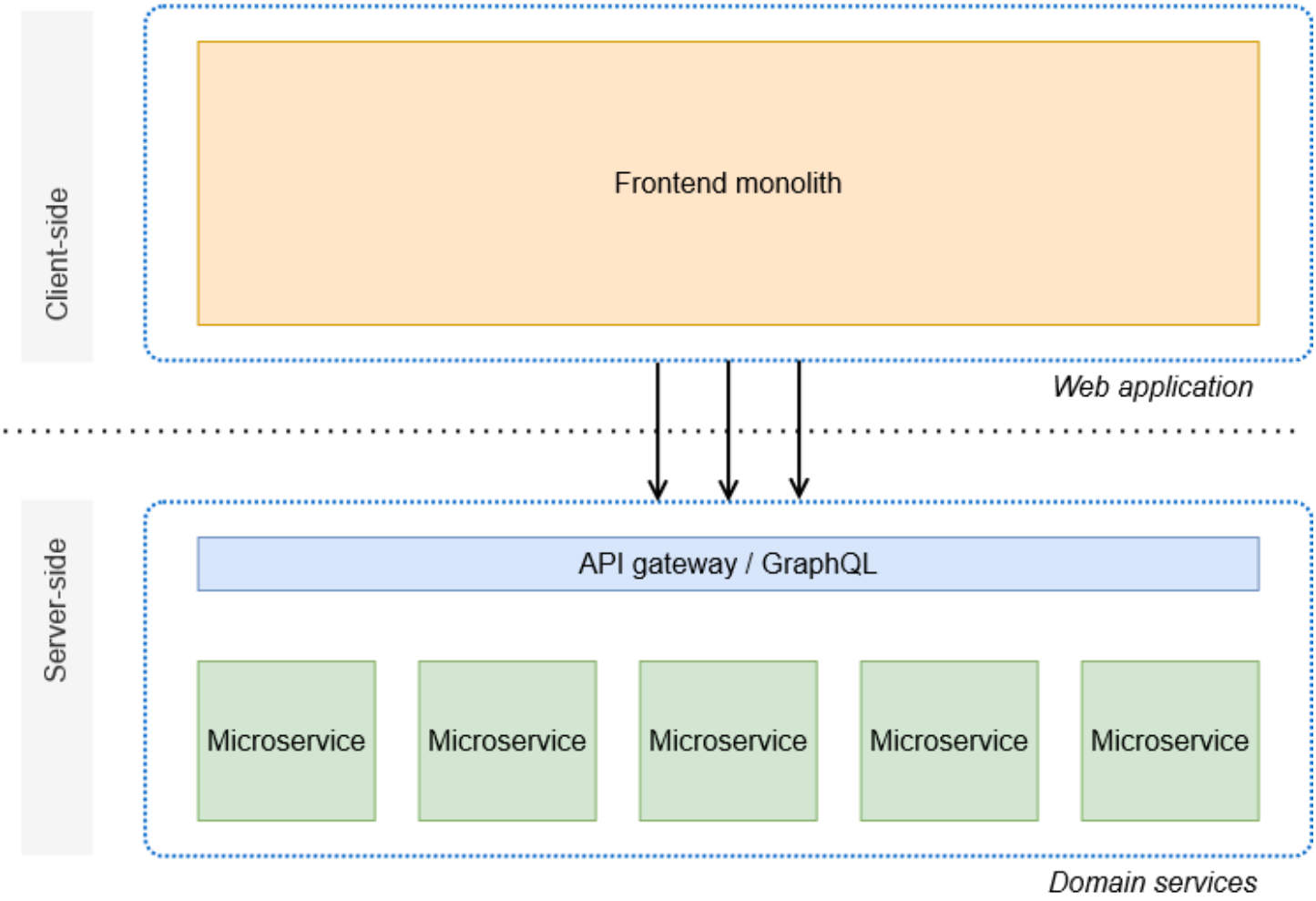
Descripción general de

Las microinterfaces son una arquitectura basada en la descomposición de las interfaces de las aplicaciones en artefactos desarrollados e implementados de forma independiente. Al dividir las interfaces de gran tamaño en artefactos de software autónomos, se puede encapsular la lógica empresarial y reducir las dependencias. Esto permite una entrega más rápida y frecuente de los incrementos de productos.

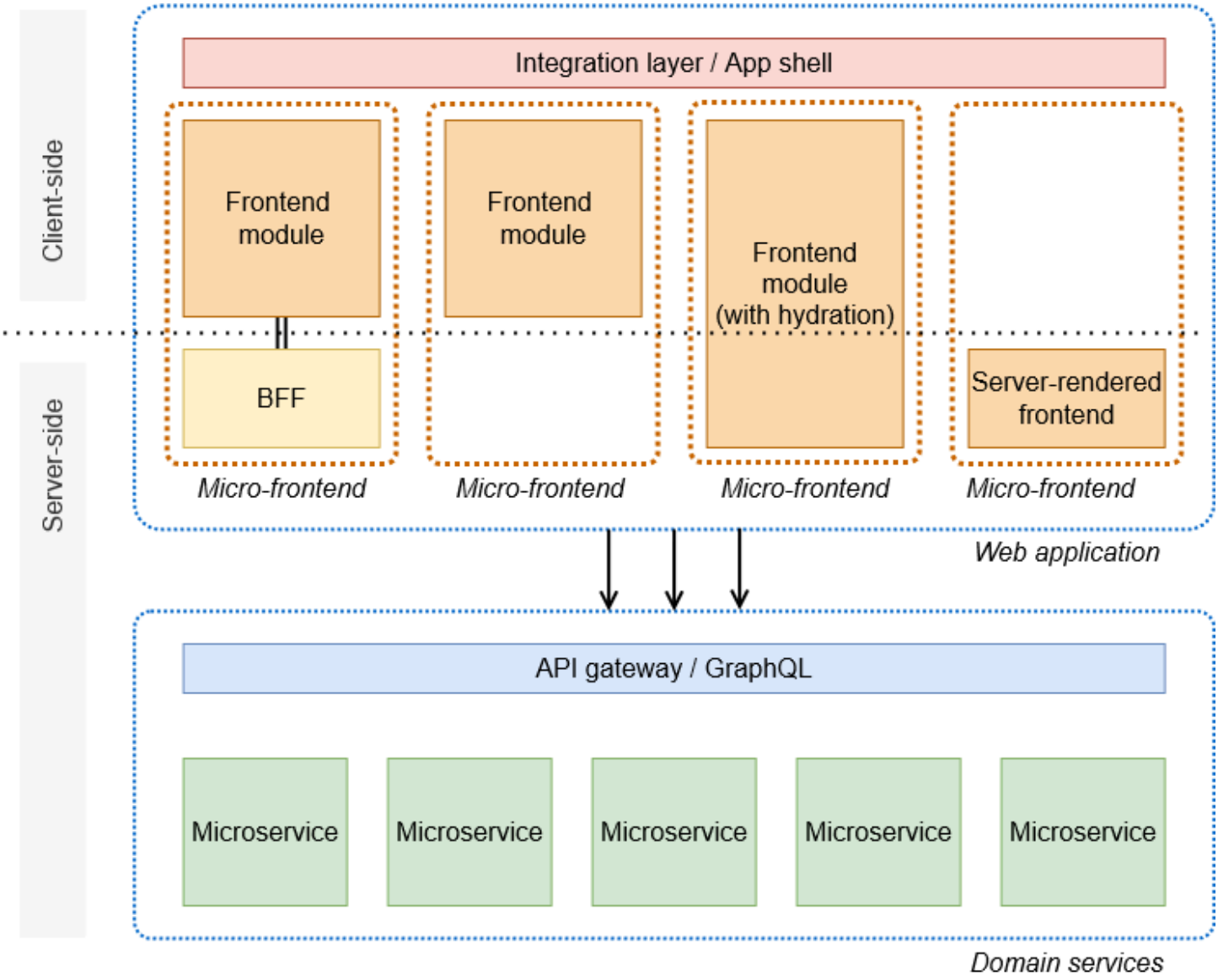
Las microinterfaces son similares a los microservicios. De hecho, el término microfrontend se deriva del término microservicio y pretende transmitir la noción de microservicio como frontend. Si bien una arquitectura de microservicios suele combinar un sistema distribuido en el backend con un frontend monolítico, los microfrontend son servicios frontend distribuidos autónomos. Estos servicios se pueden configurar de dos maneras:

- Solo para front-end, se integra con una capa de API compartida detrás de la cual se ejecuta una arquitectura de microservicios
- Completo, lo que significa que cada microinterfaz tiene su propia implementación de backend.

El siguiente diagrama muestra una arquitectura de microservicios tradicional, con un monolito de interfaz que utiliza una puerta de enlace API para conectarse a los microservicios de backend.



El siguiente diagrama muestra una arquitectura de microinterfaz con diferentes implementaciones de microservicios.



Como se muestra en el diagrama anterior, puede utilizar microinterfaces con arquitecturas de renderización del lado del cliente o del lado del servidor:

- Las microinterfaces renderizadas del lado del cliente pueden consumir directamente las exposiciones de una API APIs Gateway centralizada.
- El equipo puede crear un backend-for-frontend (BFF) dentro de un contexto limitado para reducir la conversación de la interfaz hacia el. APIs
- Por el lado del servidor, las microinterfaces se pueden expresar con un enfoque del lado del servidor y, por el lado del cliente, mediante una técnica llamada hidratación. Cuando el navegador renderiza una página, la asociada JavaScript se hidrata para permitir la interacción con los elementos de la interfaz de usuario, como hacer clic en un botón.

- Las microinterfaces pueden renderizarse en el backend y utilizar hipervínculos para dirigirse a una nueva parte de un sitio web.

Las microinterfaces son ideales para las organizaciones que desean hacer lo siguiente:

- Amplíe con varios equipos que trabajan en el mismo proyecto.
- Adopte la descentralización de la toma de decisiones, lo que permitirá a los desarrolladores innovar dentro de los límites de los sistemas identificados.

Este enfoque reduce significativamente la carga cognitiva de los equipos, ya que pasan a ser responsables de partes específicas del sistema. Aumenta la agilidad empresarial porque se pueden realizar modificaciones en una parte del sistema sin interrumpir el resto.

Las microinterfaces son un enfoque arquitectónico distinto. Si bien existen diferentes formas de crear microinterfaces, todas tienen características comunes:

- Una arquitectura de microfrontend se compone de múltiples elementos independientes. La estructura es similar a la modularización que ocurre con los microservicios en el backend.
- Una microinterfaz es completamente responsable de la implementación de la interfaz dentro de su contexto limitado, que comprende lo siguiente:
  - Interfaz de usuario
  - Datos
  - Estado o sesión
  - Lógica empresarial
  - Flujo

Un contexto acotado es un sistema internamente coherente con límites cuidadosamente diseñados que median lo que puede entrar y salir. Una microinterfaz debe compartir la menor cantidad posible de datos y lógica empresarial con otras microinterfaces. Dondequiera que sea necesario compartir, se lleva a cabo a través de interfaces claramente definidas, como eventos personalizados o transmisiones reactivas. Sin embargo, cuando se trata de cuestiones transversales, como un sistema de diseño o el registro de bibliotecas, el intercambio intencional es bienvenido.

Un patrón recomendado es crear microinterfaces mediante el uso de equipos multifuncionales. Esto significa que cada microinterfaz es desarrollada por el mismo equipo que trabaja desde

el backend hasta el frontend. La propiedad del equipo es crucial, desde la codificación hasta la operacionalización del sistema en producción.

Esta guía no pretende recomendar un enfoque en particular. En cambio, analiza diferentes patrones, mejores prácticas, compensaciones y consideraciones arquitectónicas y organizativas.

Conceptos fundamentales

La arquitectura microfrontend se inspira en gran medida en tres conceptos arquitectónicos anteriores:

- El diseño basado en dominios es el modelo mental para estructurar aplicaciones complejas en dominios coherentes.
- Los sistemas distribuidos son un enfoque para crear aplicaciones como subsistemas débilmente acoplados que se desarrollan de forma independiente y se ejecutan en su propia infraestructura dedicada.
- La computación en la nube es un enfoque para ejecutar la infraestructura de TI como servicios con un pay-as-you-go modelo.

Diseño basado en el dominio

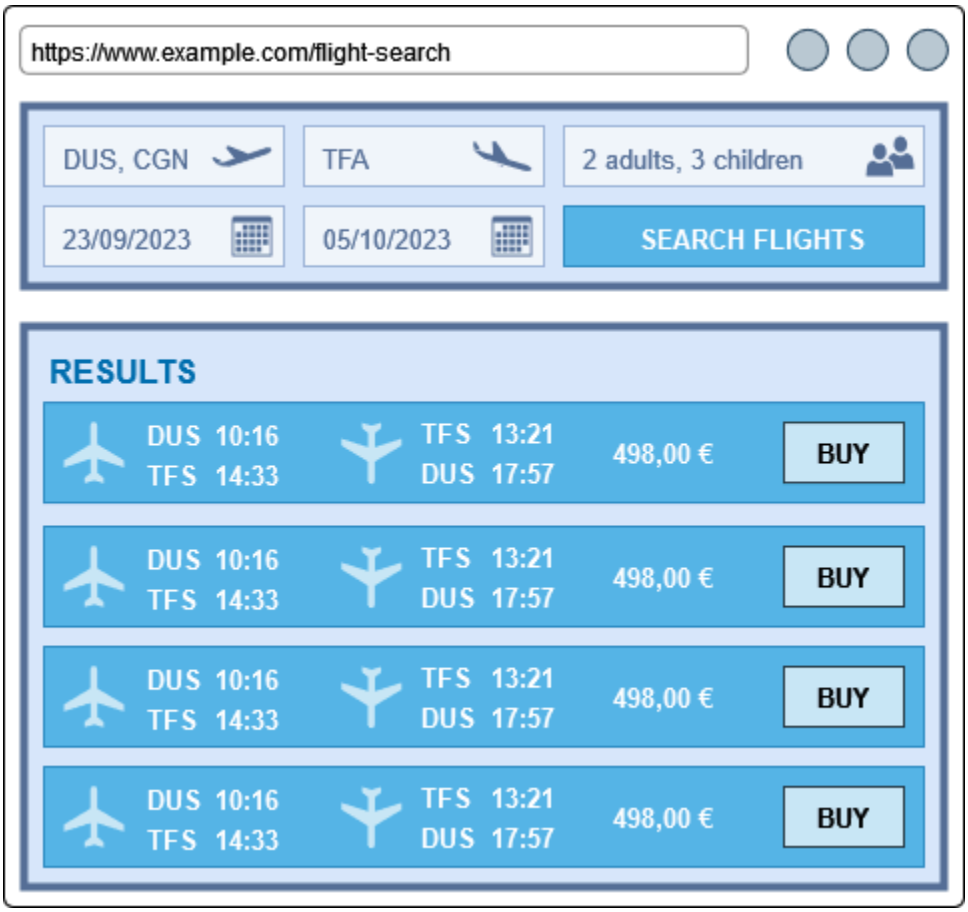
El diseño impulsado por dominios (DDD) es un paradigma desarrollado por Eric Evans. En su libro de 2003 Domain-Driven Design: Tackling Complexity in the Heart of Software, Evans postula que el desarrollo de software debe estar impulsado por preocupaciones empresariales y no por cuestiones técnicas. Evans propone que los proyectos de TI desarrollen primero un lenguaje ubicuo que ayude a los expertos técnicos y del campo a encontrar un entendimiento común. Basándose en ese lenguaje, pueden formular un modelo de la realidad empresarial que se comprenda mutuamente.

Por obvio que sea ese enfoque, muchos proyectos de software sufren una desconexión entre la empresa y la TI. Esas desconexiones suelen provocar importantes malentendidos, que se traducen en sobrecostos presupuestarios, en una disminución de la calidad o en el fracaso de los proyectos.

Evans introduce muchos otros términos importantes, uno de los cuales es el contexto limitado. Un contexto limitado es un segmento autónomo de una gran aplicación de TI que contiene la solución o implementación para exactamente una empresa empresarial. Una aplicación de gran tamaño constará de varios contextos acotados que se acoplan de forma flexible mediante patrones de integración. Esos contextos acotados pueden incluso tener sus propios dialectos del idioma ubicuo. Por ejemplo, un usuario en el contexto de pagos de una aplicación podría tener aspectos diferentes a los de un usuario en el contexto de la entrega, ya que la noción de envío sería irrelevante durante el pago.

Evans no define qué tan pequeño o grande debe ser un contexto acotado. El tamaño lo determina el proyecto de software y puede evolucionar con el tiempo. Los buenos indicadores de los límites de un contexto son el grado de cohesión entre las entidades (objetos de dominio) y la lógica empresarial.

En el contexto de las microinterfaces, el diseño basado en el dominio puede ilustrarse con el ejemplo de una página web compleja, como una página de reserva de vuelos.



En esta página, los componentes principales son un formulario de búsqueda, un panel de filtros y la lista de resultados. Para identificar los límites, debe identificar los contextos funcionales independientes. Además, tenga en cuenta los aspectos no funcionales, como la reutilización, el rendimiento y la seguridad. El indicador más importante de que «las cosas van juntas» son sus patrones de comunicación. Si algunos elementos de una arquitectura deben comunicarse con frecuencia e intercambiar información compleja, es probable que compartan el mismo contexto limitado.

Los elementos individuales de la interfaz de usuario, como los botones, no son contextos acotados, ya que no son independientes desde el punto de vista funcional. Además, la página completa no es adecuada para un contexto limitado, ya que se puede dividir en contextos independientes más pequeños. Un enfoque razonable consiste en tratar el formulario de búsqueda como un contexto acotado y tratar la lista de resultados como el segundo contexto acotado. Cada uno de estos dos contextos acotados ahora se puede implementar como una microinterfaz independiente.

Sistemas distribuidos

Para facilitar el mantenimiento y respaldar la capacidad de evolución, la mayoría de las soluciones de TI no triviales son modulares. En este caso, modular significa que los sistemas de TI constan de componentes básicos identificables que se disocian mediante interfaces para lograr separar las preocupaciones.

Además de ser modulares, los sistemas distribuidos deben ser sistemas independientes por derecho propio. En un sistema meramente modular, cada módulo está encapsulado idealmente y expone sus funciones a través de interfaces, pero no se puede implementar de forma independiente ni siquiera funcionar por sí solo. Además, los módulos suelen seguir el mismo ciclo de vida que otros módulos que forman parte del mismo sistema. Por otro lado, cada uno de los componentes básicos de un sistema distribuido tiene sus propios ciclos de vida. Al aplicar el paradigma del diseño basado en el dominio, cada componente básico se dirige a un dominio o subdominio empresarial y vive en su propio contexto limitado.

Cuando los sistemas distribuidos interactúan durante el tiempo de construcción, un enfoque común es desarrollar mecanismos para identificar los problemas rápidamente. Por ejemplo, puede adoptar lenguajes mecanografiados e invertir mucho en pruebas unitarias. Varios equipos pueden colaborar en el desarrollo y el mantenimiento de los módulos, que suelen distribuirse como bibliotecas para que los sistemas los consuman con herramientas como npm, Apache Maven y pip NuGet.

Durante el tiempo de ejecución, los sistemas distribuidos que interactúan suelen ser propiedad de equipos individuales. El consumo de dependencias provoca complejidad operativa debido a la gestión de errores, el equilibrio del rendimiento y la seguridad. Las inversiones en las pruebas de integración y la observabilidad son fundamentales para reducir los riesgos.

Los ejemplos más populares de sistemas distribuidos en la actualidad son los microservicios. En las arquitecturas de microservicios, los servicios de backend dependen del dominio (y no están motivados por cuestiones técnicas como la interfaz de usuario o la autenticación) y son propiedad de equipos autónomos. Las microinterfaces comparten los mismos principios, lo que amplía el alcance de la solución a la interfaz.

Computación en la nube

La computación en la nube es una forma de adquirir infraestructura de TI como servicios con un pay-as-you-go modelo, en lugar de construir sus propios centros de datos y comprar hardware para operarlos in situ. La computación en nube ofrece varias ventajas:

- Su organización obtiene una agilidad empresarial significativa al poder experimentar con nuevas tecnologías sin tener que asumir grandes compromisos financieros a largo plazo por adelantado.
- Al utilizar un proveedor de servicios en la nube, por ejemplo AWS, su organización puede acceder a una amplia cartera de servicios que requieren poco mantenimiento y son altamente integrables (como pasarelas de API, bases de datos, organización de contenedores y capacidades de nube). El acceso a estos servicios permite a su personal centrarse en el trabajo que diferencia a su organización de la competencia.
- Cuando su organización esté lista para implementar una solución a nivel mundial, podrá implementarla en la infraestructura de la nube de todo el mundo.

La computación en la nube es compatible con las microinterfaces al proporcionar una infraestructura altamente gestionada. Esto facilita end-to-end la propiedad para los equipos multifuncionales. Si bien el equipo debe tener sólidos conocimientos sobre operaciones, las tareas manuales de aprovisionamiento de la infraestructura, actualizaciones del sistema operativo y redes serían una distracción.

Como las microinterfaces se encuentran en contextos limitados, los equipos pueden elegir el servicio más adecuado para ejecutarlas. Por ejemplo, los equipos pueden elegir entre funciones de nube y contenedores para el procesamiento, y pueden elegir entre diferentes tipos de bases de datos SQL y NoSQL o cachés en memoria. Los equipos pueden incluso crear sus microinterfaces a partir de un conjunto de herramientas altamente integrado, por ejemplo AWS Amplify, que incluye componentes básicos preconfigurados para una infraestructura sin servidor.

Comparación de microinterfaces con arquitecturas alternativas

Como ocurre con todas las estrategias de arquitectura, la decisión de adoptar microinterfaces debe basarse en criterios de evaluación que se guíen por los principios de la organización. Las microinterfaces tienen ventajas y desventajas. Si su organización decide utilizar microinterfaces, debe contar con estrategias para abordar los desafíos de los sistemas distribuidos

Al elegir una arquitectura de aplicaciones, las alternativas más populares a las microinterfaces son los monolitos, las aplicaciones de n niveles y los microservicios en combinación con una interfaz de aplicación de una sola página (SPA). Todos estos enfoques son válidos y cada uno de ellos tiene ventajas y desventajas.

Monolitos

Una aplicación pequeña que no necesite cambios frecuentes se puede entregar muy rápidamente como un monolito. Incluso en situaciones en las que se espera un crecimiento significativo, un monolito es un primer paso natural. Más adelante, el monolito se puede retirar o refactorizar para crear una estructura más flexible. Al empezar con un monolito, su organización puede lanzarse al mercado, obtener los comentarios de los clientes y mejorar el producto con mayor rapidez.

Sin embargo, las aplicaciones monolíticas tienden a degradarse si no se mantienen cuidadosamente o a medida que la base de código aumenta de tamaño con el tiempo. Cuando varios equipos contribuyen de manera significativa a la misma base de código, rara vez todos contribuyen a su mantenimiento y operaciones. Esto se traduce en un desequilibrio de responsabilidades, lo que repercute en la velocidad y provoca ineficiencias. Al mismo tiempo, el acoplamiento inadvertido entre los módulos de un monolito provoca efectos secundarios no deseados a medida que la base de código evoluciona. Estos efectos secundarios pueden provocar averías e interrupciones.

Aplicaciones de nivel N

Una aplicación más compleja que tenga un ritmo de evolución relativamente estático se puede crear como una arquitectura de tres niveles (presentación, aplicación, datos), con una capa REST o GraphQL entre el frontend y el backend. Esto es mucho más flexible y los equipos de los diferentes niveles pueden desarrollarse de forma independiente hasta cierto punto. La desventaja de una aplicación de n niveles es que es mucho más difícil implementar la funcionalidad. El frontend y el

backend están disociados mediante un contrato de API, por lo que los cambios más importantes se deben implementar juntos o se debe versionar la API.

Considere el siguiente escenario común: si el lanzamiento de una nueva función requiere un cambio en el esquema de datos, los propietarios del producto pueden tardar días en ponerse de acuerdo sobre un conjunto de funcionalidades con el equipo de front-end. A continuación, el equipo de frontend pedirá al equipo de backend que desarrolle y publique la funcionalidad por su parte. El equipo de backend trabajará con los propietarios de los datos para publicar una actualización del esquema de la base de datos. A continuación, el equipo de backend lanzará una nueva versión de la API para que el equipo de front-end pueda desarrollar y publicar sus cambios. En este escenario, la propagación de todos los cambios a la fase de producción puede llevar semanas o incluso meses, ya que cada equipo tiene sus propias tareas pendientes, prioridades y mecanismos en relación con el desarrollo, las pruebas y la publicación de los cambios.

Microservicios

En una arquitectura de microservicios, el backend se divide en pequeños servicios, cada uno de los cuales aborda un problema empresarial concreto dentro de un contexto limitado. Cada microservicio también está fuertemente disociado de otros servicios al exponer un contrato de interfaz claramente definido.

Vale la pena mencionar que los contextos limitados y los contratos de interfaz también deberían existir en monolitos bien diseñados y arquitecturas de n niveles. Sin embargo, en una arquitectura de microservicios, la comunicación se produce a través de la red, normalmente a través del protocolo HTTP, y los servicios tienen una infraestructura de tiempo de ejecución dedicada. Esto permite el desarrollo, la entrega y el funcionamiento independientes de cada servicio de backend.

Elegir el enfoque que mejor se adapte a sus necesidades

Los monolitos y las arquitecturas de n niveles agrupan varios problemas de dominio en un solo artefacto técnico. Esto facilita la gestión de aspectos como las dependencias y el flujo interno de datos, pero dificulta la entrega de nuevas funcionalidades. Para mantener una base de código coherente, un equipo suele invertir tiempo en la refactorización y la desvinculación debido a la gran cantidad de código que tienen que gestionar.

Es posible que las aplicaciones desarrolladas por unos pocos equipos no necesiten la complejidad adicional que conlleva el paso a las microinterfaces. Esto es especialmente cierto si los equipos no

están pagando las penalizaciones que supone el elevado número de conexiones y los largos plazos de entrega de los cambios.

En resumen, las arquitecturas más complejas y distribuidas suelen ser la elección correcta para aplicaciones complejas y de rápida evolución. En el caso de las aplicaciones pequeñas y medianas, una arquitectura distribuida no es necesariamente superior a una monolítica, especialmente si la aplicación no va a evolucionar drásticamente en un corto período de tiempo.

Decisiones arquitectónicas en microfrontend

Los equipos que apliquen un patrón de arquitectura de microinterfaz para sus aplicaciones deben tomar varias decisiones sobre la arquitectura desde el principio:

- Identificación de las microinterfaces y definición de los límites
- Composición de páginas y vistas con microinterfaces
- Enrutamiento, administración del estado y comunicación a través de microinterfaces
- Gestionar las dependencias para resolver problemas transversales

Las siguientes secciones tratan estos temas con mayor profundidad.

Al tomar decisiones de arquitectura, es esencial contar con las métricas correctas y comprender los patrones de uso, las características de las aplicaciones y las ventajas y desventajas. Por ejemplo, un sitio de comercio electrónico tiene características y patrones de uso diferentes en comparación con una herramienta de edición de vídeo o con los paneles de observabilidad.

Las aplicaciones orientadas al público con mucho tráfico y sesiones de corta duración se pueden optimizar para las métricas de carga inicial de la página, como Time to Interactive (TTI) y First Contentful Paint (FCP). Por el contrario, una aplicación en la que los usuarios inicien sesión al principio del día y con la que sigan interactuando durante todo el día podría estar optimizada para la experiencia dentro de la aplicación. El equipo de aplicaciones podría optimizar la métrica del retraso de la primera entrada (FID) después de cada navegación en lugar de utilizar la carga inicial de la página.

Los sitios web públicos deben adaptarse a los distintos entornos de navegación. Las aplicaciones empresariales con limitaciones conocidas en el entorno del cliente pueden optimizar la composición de su microinterfaz en función de esas limitaciones.

No existe una única opción correcta para las decisiones de arquitectura. Comprenda las ventajas y desventajas, el contexto en el que opera la empresa, los patrones de uso y las métricas para guiar las decisiones adecuadas para cada aplicación individual.

Identificar los límites de la microinterfaz

Para mejorar la autonomía del equipo, las capacidades empresariales que ofrece una aplicación se pueden descomponer en varias microinterfaces con una dependencia mínima entre sí.

Siguiendo la metodología DDD descrita anteriormente, los equipos pueden dividir el dominio de una aplicación en subdominios empresariales y contextos acotados. De este modo, los equipos autónomos pueden hacerse cargo de la funcionalidad de sus contextos acotados y ofrecer esos contextos como microinterfaces.

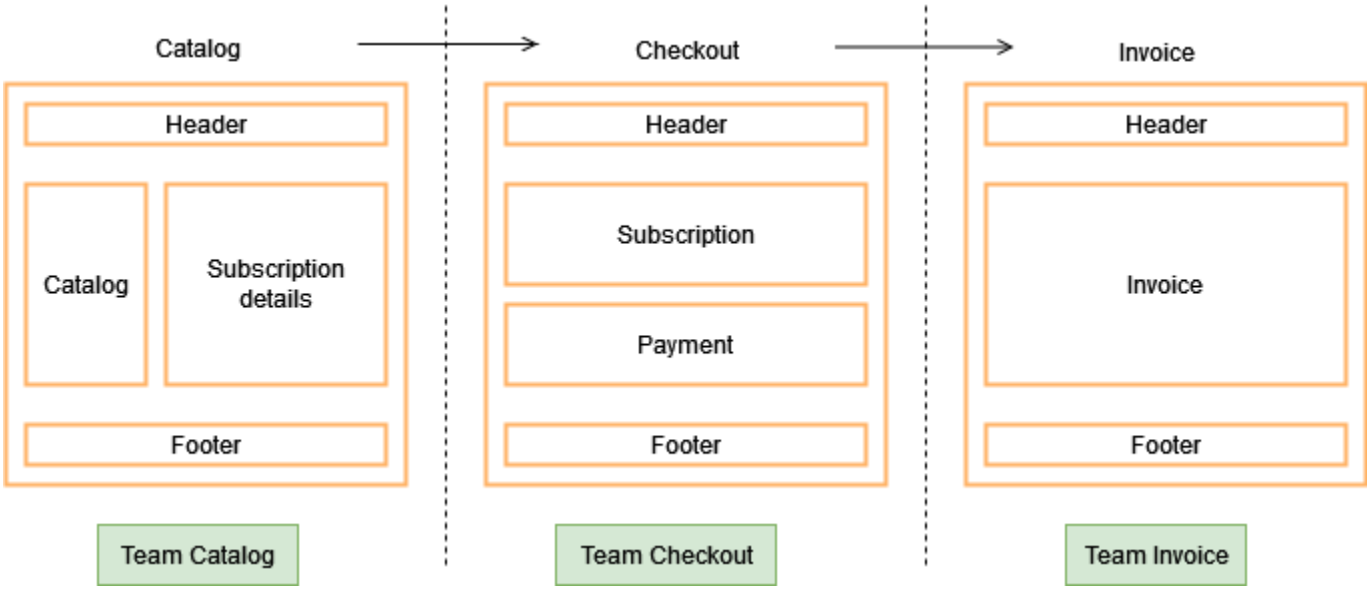
Un contexto acotado bien definido debería minimizar la superposición funcional y la necesidad de una comunicación en tiempo de ejecución entre contextos. La comunicación requerida se puede implementar con métodos basados en eventos. Esto no es diferente de la arquitectura basada en eventos para el desarrollo de microservicios.

Una aplicación bien diseñada también debería respaldar la entrega de futuras extensiones por parte de nuevos equipos para proporcionar una experiencia coherente a los clientes.

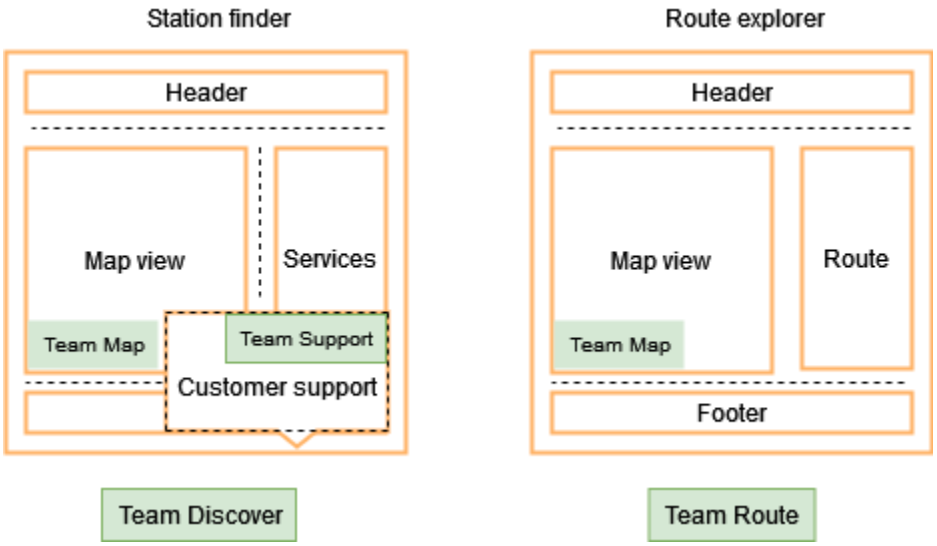
¿Cómo dividir una aplicación monolítica en microinterfaces

La sección de descripción general incluía un ejemplo de cómo identificar contextos funcionales independientes en una página web. Surgen varios patrones para dividir la funcionalidad en la interfaz de usuario.

Por ejemplo, cuando los dominios empresariales forman etapas del recorrido del usuario, se puede aplicar una división vertical en la interfaz, en la que una colección de vistas del recorrido del usuario se presenta en forma de microinterfaces. El siguiente diagrama muestra una división vertical, en la que los pasos de catálogo, pago y facturación los llevan a cabo equipos separados como microinterfaces independientes.



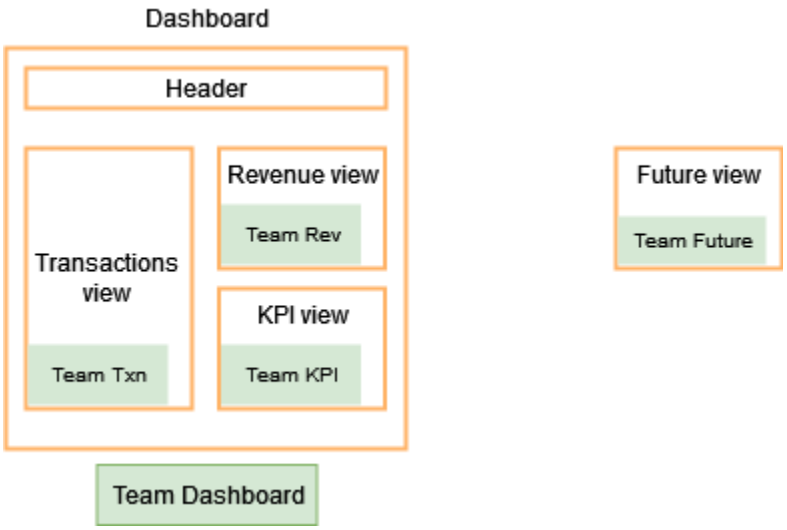
Para algunas aplicaciones, la división vertical por sí sola puede no ser suficiente. Por ejemplo, es posible que sea necesario proporcionar alguna funcionalidad en varias vistas. Para estas aplicaciones, puede aplicar una división mixta. El siguiente diagrama muestra una solución de división mixta en la que las microinterfaces de Station Finder y Route Explorer utilizan la función de visualización de mapas.



Las aplicaciones tipo portal o panel de control suelen reunir las capacidades de la interfaz de usuario en una sola vista. En este tipo de aplicaciones, cada widget se puede entregar como una microinterfaz, y la aplicación de alojamiento define las restricciones e interfaces que deben implementar las microinterfaces.

Este enfoque proporciona un mecanismo para que las microinterfaces resuelvan problemas como el tamaño de las ventanas gráficas, los proveedores de autenticación, los ajustes de configuración y los metadatos. Estos tipos de aplicaciones optimizan la extensibilidad. Los nuevos equipos pueden desarrollar nuevas funciones para ampliar las capacidades del panel de control.

El siguiente diagrama muestra una aplicación de panel desarrollada por tres equipos individuales que forman parte de Team Dashboard.



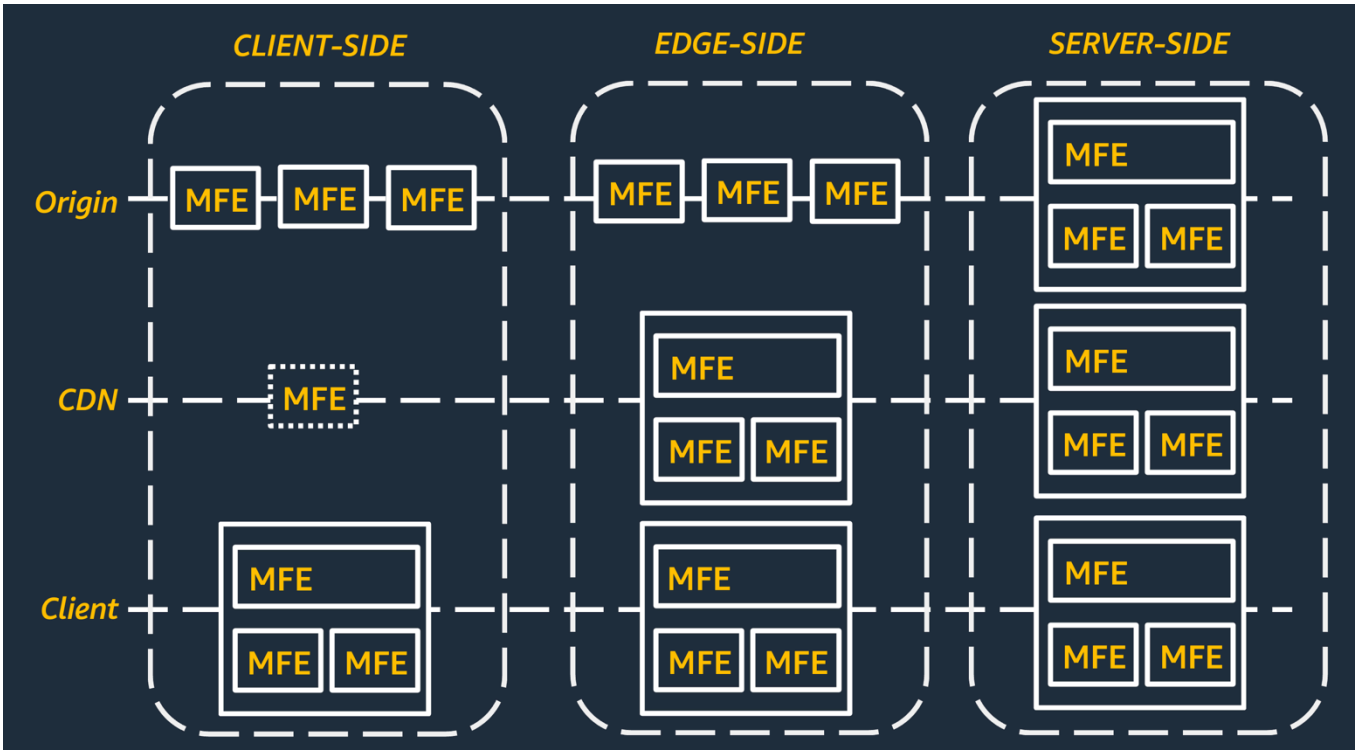
En el diagrama, la vista futura representa las nuevas funciones desarrolladas por los nuevos equipos para escalar el panel de control del equipo y las capacidades del panel de control.

Las aplicaciones de portal y panel de control suelen componer la funcionalidad mediante una división mixta en la interfaz de usuario. Las microinterfaces se pueden configurar con ajustes bien definidos, que incluyen restricciones de posición y tamaño.

Composición de páginas y vistas con microinterfaces

Puede crear vistas de una aplicación con una composición del lado del cliente, una composición del lado del borde y una composición del lado del servidor. Los patrones de composición tienen características diferentes en cuanto a las habilidades de equipo necesarias, la tolerancia a los errores, el rendimiento y el comportamiento de la memoria caché.

El siguiente diagrama muestra cómo se produce la composición en las capas del lado del cliente, del lado perimetral y del lado del servidor de una arquitectura de microinterfaz.



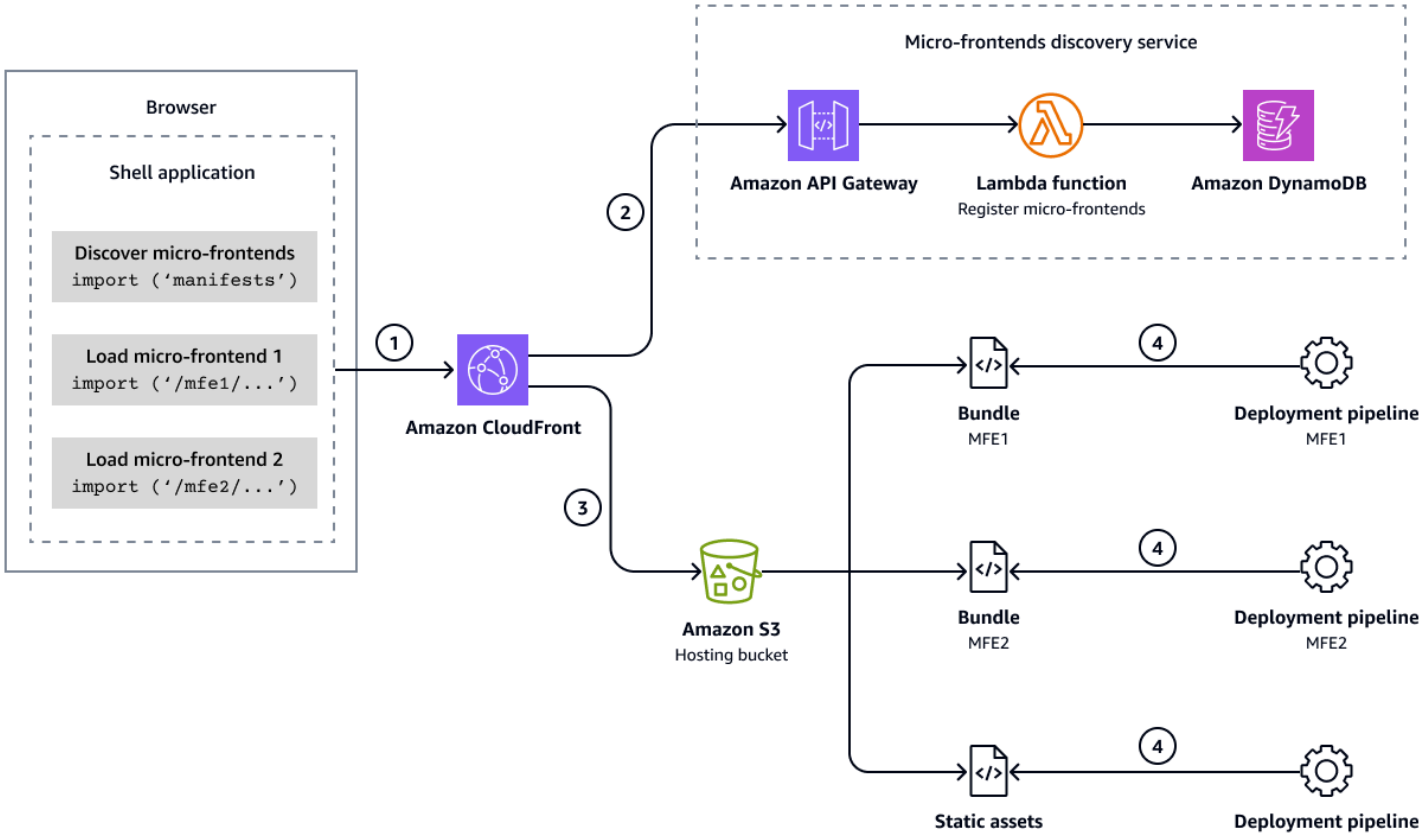
Las capas del lado del cliente, del lado perimetral y del lado del servidor se analizan en las siguientes secciones.

Composición del cliente

Cargue y añada de forma dinámica las microinterfaces como fragmentos del Document Object Model (DOM) en el cliente (navegador o vista web móvil). Los artefactos de la microinterfaz, como JavaScript los archivos CSS, se pueden cargar desde las redes de entrega de contenido () CDNs para reducir la latencia. La composición del lado del cliente requiere lo siguiente:

- Un equipo que posea y mantenga una aplicación shell o un marco de microinterfaz para poder descubrir, cargar y renderizar los componentes de la microinterfaz en tiempo de ejecución en el navegador
- Altos niveles de habilidad en tecnologías de interfaz, como HTML y CSS, y JavaScript un conocimiento profundo de los entornos de los navegadores
- Optimización de la cantidad de JavaScript carga de una página y disciplina para evitar conflictos globales entre los espacios de nombres

El siguiente diagrama muestra un ejemplo de AWS arquitectura para la composición del lado del cliente sin servidor.



La composición desde el lado del cliente se realiza en el entorno del navegador a través de una aplicación shell. El diagrama muestra los siguientes detalles:

1. Una vez cargada la aplicación shell, hace una solicitud inicial a Amazon para que descubra las microinterfaces CloudFront que se van a cargar a través de un punto final de manifiesto.
2. Los manifiestos contienen información sobre cada microinterfaz (por ejemplo, nombre, URL, versión y comportamiento alternativo). Los manifiestos los proporciona el servicio de detección de microinterfaces. En el diagrama, este servicio de detección está representado por Amazon API Gateway, una AWS Lambda función, y Amazon DynamoDB. La aplicación shell utiliza la información del manifiesto para solicitar a las microinterfaces individuales que redacten la página con un diseño determinado.
3. Cada paquete de microinterfaces se compone de archivos estáticos (como CSS JavaScript y HTML). Los archivos se alojan en un depósito de Amazon Simple Storage Service (Amazon S3) y se sirven allí. CloudFront
4. Los equipos pueden implementar nuevas versiones de sus microinterfaces y actualizar la información del manifiesto mediante procesos de implementación de los que son propietarios.

Composición desde los bordes

Utilice técnicas de transclusión, como Edge Side Includes (ESI) o Server Side Includes (SSI), compatibles con algunos servidores, CDNs y utilice proxies frente a los servidores de origen para componer una página antes de enviarla por cable a los clientes. ESI requiere lo siguiente:

- Una CDN con capacidad ESI o un despliegue de proxy frente a las microinterfaces del lado del servidor. Las implementaciones de proxy, como Varnish y NGINX HAProxy, admiten SSI.
- Comprensión del uso y las limitaciones de las implementaciones de ESI y SSI.

Los equipos que inician nuevas aplicaciones no suelen elegir la composición lateral como patrón de composición. Sin embargo, este patrón podría proporcionar una vía para las aplicaciones antiguas que se basan en la transclusión.

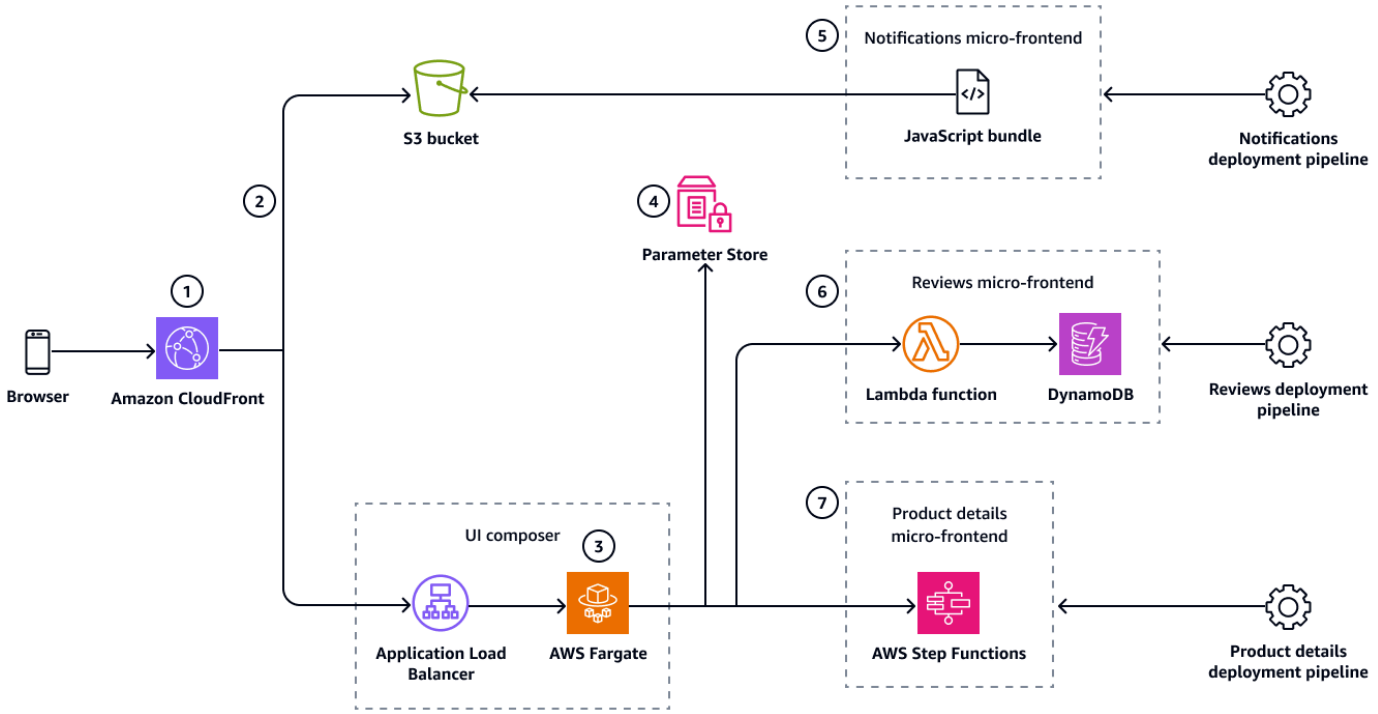
Composición del servidor

Utilice los servidores de origen para componer las páginas antes de que se almacenen en caché en el borde. Esto se puede hacer con tecnologías tradicionales, como PHP, Jakarta Server Pages (JSP) o bibliotecas de plantillas, para componer las páginas incluyendo fragmentos de microinterfaces. También puede utilizar JavaScript marcos, como Next.js, que se ejecutan en el servidor para crear páginas en el servidor con renderización del lado del servidor (SSR).

Una vez renderizadas las páginas en el servidor, se pueden almacenar en CDNs caché para reducir la latencia. Cuando se implementan nuevas versiones de microinterfaces, las páginas deben volver a renderizarse y la memoria caché debe actualizarse para ofrecer las versiones más recientes a los clientes.

La composición del lado del servidor requiere una comprensión profunda del entorno del servidor para establecer patrones de implementación, descubrir las microinterfaces del lado del servidor y administrar la memoria caché.

El siguiente diagrama muestra la composición del lado del servidor.



El diagrama incluye los siguientes componentes y procesos:

1. Amazon CloudFront proporciona un punto de entrada único a la aplicación. La distribución tiene dos orígenes: el primero para los archivos estáticos y el segundo para el compositor de la interfaz de usuario.
2. Los archivos estáticos se alojan en un bucket de Amazon S3. Los consumen el navegador y el compositor de la interfaz de usuario para las plantillas HTML.
3. El compositor de la interfaz de usuario se ejecuta en un clúster de contenedores en AWS Fargate. Con una solución contenerizada, puedes usar las capacidades de streaming y el renderizado multiproceso si es necesario.
4. Parameter Store, una capacidad de AWS Systems Manager, se utiliza como un sistema básico de descubrimiento de microinterfaces. Esta capacidad proporciona un almacén de valores clave que utiliza el compositor de la interfaz de usuario para recuperar los puntos finales de la microinterfaz y consumirlos.
5. La microinterfaz de notificaciones almacena el paquete optimizado en el depósito de S3. JavaScript Esto se refleja en el cliente porque debe reaccionar ante las interacciones de los usuarios.
6. La microinterfaz de reseñas está compuesta por una función Lambda y las reseñas de los usuarios se almacenan en DynamoDB. La microinterfaz de reseñas se renderiza completamente en el servidor y genera un fragmento HTML.

7. La microinterfaz de detalles del producto es una microinterfaz de bajo código que utiliza. AWS Step Functions El Express Workflow se puede invocar de forma sincrónica y contiene la lógica para renderizar el fragmento HTML y una capa de almacenamiento en caché.

Para obtener más información sobre la composición del lado del servidor, consulte la entrada del blog Server-side side rendering micro-frontends: the architecture.

Enrutamiento y comunicación entre microinterfaces

Las opciones de enrutamiento dependen del enfoque de composición. La comunicación se puede optimizar reduciendo el acoplamiento entre los componentes de la interfaz.

Enrutamiento

Las aplicaciones que utilizan la composición del lado del cliente con división vertical pueden utilizar el enrutamiento del lado del servidor (aplicación multipágina) o el enrutamiento del lado del cliente (aplicación de una sola página). Si utilizan una división mixta para la composición de la interfaz de usuario, el enrutamiento del lado del cliente es necesario para admitir jerarquías de enrutamiento más profundas de las microinterfaces de una página.

Las aplicaciones que utilizan la composición del lado del borde y la composición del lado del servidor se alinean mejor con el enrutamiento del lado del servidor o con la computación perimetral, como Lambda @Edge con Amazon. CloudFront

Comunicación entre microinterfaces

Con las arquitecturas de microfrontend, recomendamos reducir el acoplamiento entre los componentes de la interfaz. Un enfoque para reducir el acoplamiento consiste en pasar de las llamadas a funciones sincrónicas a la mensajería asíncrona.

Los tiempos de ejecución del navegador y las interacciones de los usuarios son asíncronos por naturaleza. Los eventos se pueden intercambiar entre productores y consumidores a través de mensajes. Los eventos proporcionan una interfaz bien definida para la comunicación a través de microinterfaces.

Si sigues las prácticas de la DDD para identificar tus contextos acotados para las microinterfaces, el siguiente paso es identificar los eventos que deben comunicarse más allá de los límites.

El mecanismo de mensajería para los eventos puede ser eventos DOM nativos (`CustomEvents`), JavaScript emisores de eventos o bibliotecas de transmisiones reactivas proporcionadas por los equipos de la plataforma. Las microinterfaces publican eventos y se suscriben a eventos relevantes para su contexto limitado. Con este método, los editores y los suscriptores no necesitan conocerse entre sí. El contrato es la definición del evento. Para obtener una representación visual de esto, consulte la sección Cómo comunicarse con los eventos del diagrama Contexto limitado con arquitecturas de eventos.

Gestionar las dependencias para problemas transversales

La gestión consciente de las dependencias es crucial para el éxito de una arquitectura distribuida, como las microinterfaces. La gestión de las dependencias es una de las partes más desafiantes del desarrollo de las microinterfaces.

En una arquitectura de microinterfaz, dos aspectos importantes de la gestión de las dependencias son la reducción del rendimiento que supone la transferencia de grandes fragmentos de código al cliente y la sobrecarga de los recursos informáticos. Lo ideal es que su organización determine cómo se mantienen las dependencias en una arquitectura de interfaz distribuida.

Tres estrategias viables para exigir el mantenimiento de las dependencias son no compartir nada, utilizando estándares web como la importación de mapas y la federación de módulos. Otros enfoques son contrarios a los patrones porque violan los principios básicos de las arquitecturas distribuidas.

No compartas nada, siempre que sea posible

El enfoque de no compartir nada postula que no se deben compartir en absoluto las dependencias entre artefactos de software independientes, o al menos no durante la integración o el tiempo de ejecución. Esto significa que si dos microinterfaces dependen de la misma biblioteca, cada una debe estar integrada en la biblioteca en el momento de la compilación y enviarla por separado. Además, cada microinterfaz debe validar que la biblioteca no contamine los espacios de nombres globales ni los recursos compartidos.

Esto lleva a redundancias, pero se trata de una compensación consciente con la máxima agilidad. Al no compartir dependencias de tiempo de ejecución, los equipos disponen de la máxima flexibilidad para desarrollar el software de la forma que consideren útil, siempre y cuando lo hagan dentro del ámbito de su solución y no rompan ningún contrato de interfaz.

En una plataforma en la que las microinterfaces siguen el principio de no compartir nada, es importante que las microinterfaces sean lo más ligeras posible. Requiere desarrolladores que sean expertos y diligentes a la hora de optimizar sus microinterfaces para mejorar el rendimiento y que no sacrifiquen la experiencia del usuario por la del desarrollador.

Cuando compartes código

Cuando tomas la decisión de compartir algún código, puedes compartirlo como bibliotecas o módulos de tiempo de ejecución. Por ejemplo, el equipo central de frontend ofrece bibliotecas para el consumo de microinterfaces. CDNs Los equipos de valor empresarial pueden cargar las bibliotecas en tiempo de ejecución o pueden utilizar los repositorios de paquetes para publicar sus bibliotecas. Los equipos con microinterfaz pueden desarrollar aplicaciones basadas en una versión específica de la biblioteca empaquetada en el momento de la compilación, de forma similar a las aplicaciones móviles que utilizan marcos híbridos.

Una tercera opción es utilizar un registro de paquetes privado para permitir la integración de bibliotecas comunes en el momento de la compilación. Esto reduce el riesgo de que un cambio importante en el contrato de la biblioteca provoque errores durante el tiempo de ejecución. Sin embargo, este enfoque más conservador requiere una mayor gobernanza para sincronizar todas las microinterfaces con las versiones más recientes de la biblioteca.

Para mejorar los tiempos de carga de la página, las microinterfaces pueden externalizar las dependencias de la biblioteca para cargarlas desde fragmentos en caché desde una CDN como Amazon. CloudFront

Para gestionar las dependencias del tiempo de ejecución, las microinterfaces pueden utilizar mapas de importación (o bibliotecas, por ejemplo`System.js`) para especificar desde dónde se carga cada módulo en tiempo de ejecución. La federación de módulos de webpack es otro enfoque para seleccionar una versión alojada de un módulo remoto y resolver las dependencias comunes entre microinterfaces independientes.

Otro enfoque consiste en facilitar la carga dinámica de los mapas de importación con una solicitud inicial a un punto final de detección.

Estado compartido

Para reducir el acoplamiento de las microinterfaces, es importante evitar una gestión del estado global accesible desde todas las microinterfaces desde la misma vista, similar a las arquitecturas

monolíticas. Por ejemplo, disponer de una tienda global de Redux accesible desde todas las microinterfaces aumenta el acoplamiento.

Un patrón para eliminar el estado compartido es encapsularlo en microinterfaces y comunicarse con mensajes asíncronos, como se ha explicado anteriormente.

Cuando sea absolutamente necesario, introduzca interfaces bien definidas para el estado global y opte por compartir en modo de solo lectura para evitar comportamientos inesperados:

- Cuando hay una división vertical, puede utilizar los componentes de la URL y el almacenamiento del navegador para acceder a la información del entorno anfitrión.
- Si tiene una división mixta, también puede utilizar JavaScript bibliotecas o eventos personalizados estándar del DOM, como emisores de eventos o transmisiones bidireccionales, para pasar información a las microinterfaces.

Si necesitas compartir varios datos entre las microinterfaces, te recomendamos revisar los límites de las microinterfaces. La necesidad de compartir puede deberse a la evolución empresarial o a un diseño inicial deficiente.

También es posible utilizar sesiones del lado del servidor, en las que cada microinterfaz obtiene los datos necesarios mediante un identificador de sesión. Para reducir el acoplamiento, es importante eliminar el estado compartido y mantener separados los datos de sesión específicos de la microinterfaz.

Marcos y herramientas

No faltan marcos de interfaz, como Angular y Next.js, pero la mayoría de ellos no se crean pensando en las microinterfaces. Por lo tanto, a veces faltan mecanismos para abordar los desafíos de la arquitectura microfrontend.

Consideraciones generales sobre el marco

El objetivo de esta guía no es recomendar ni comparar marcos individuales. Como a menudo se ejecutan varias microinterfaces en la misma página de aplicación web, las principales preocupaciones son la carga y el rendimiento del tiempo de ejecución. Es importante elegir un marco que presente la menor sobrecarga posible.

Los marcos se dividen en función de la capa de renderizado:

- Renderización del lado del cliente (CSR)
- Renderización del servidor (SSR)

Las arquitecturas frontend incluyen otras capacidades, como la generación de sitios estáticos (SSG). Sin embargo, el SSG se realiza solo una vez. Las microinterfaces se componen principalmente en tiempo de ejecución, por lo que las principales opciones son CSR y SSR.

Renderización del lado del cliente

Para la RSE, hay dos opciones populares:

- Estructura SPA única
- Federación de módulos

Single SPA es una opción ligera para crear microinterfaces. Resuelve los desafíos más comunes en las arquitecturas de microfrontend, como componer varias microinterfaces en la misma página y evitar conflictos de dependencia.

Module Federation comenzó como un complemento, ofrecido por webpack 5, y resuelve la gran mayoría de los desafíos de las arquitecturas de microfrontend, incluida la gestión de las dependencias entre diferentes artefactos. Module Federation 2.0 funciona de forma nativa con Rspack, webpack, esbuild y, ahora, con. JavaScript

Considere no usar ningún marco en absoluto. Los navegadores modernos, con una cuota de mercado global del 98 por ciento según caniuse.com, ofrecen funciones como elementos personalizados de forma nativa y son adecuadas para una aplicación con microinterfaces. Cuando sea necesario, combine elementos personalizados con bibliotecas ligeras para la propagación de eventos, la internacionalización u otros problemas específicos.

Renderización del lado del servidor

Por el lado del SSR, las dos opciones principales son más complicadas:

- Adopte un marco existente, como Next.js, y aplique un principio de microinterfaces que utilice la federación de módulos.
- Utilice HTML- over-the-wire para intercambiar fragmentos de HTML que representen microinterfaces y componga estos fragmentos dentro de una plantilla en tiempo de ejecución. Un ejemplo de este enfoque es Podium.

Integración de API – Backend para frontend

El patrón Backends for Frontends (BFF) se utiliza normalmente en entornos de microservicios. En el contexto de las microinterfaces, una BFF es un servicio del lado del servidor que pertenece a una microinterfaz. No todas las microinterfaces necesitan tener una mejor amiga. Sin embargo, si utilizas un BFF, debe ejecutarse dentro del mismo contexto acotado y no compartirse en otros contextos acotados.

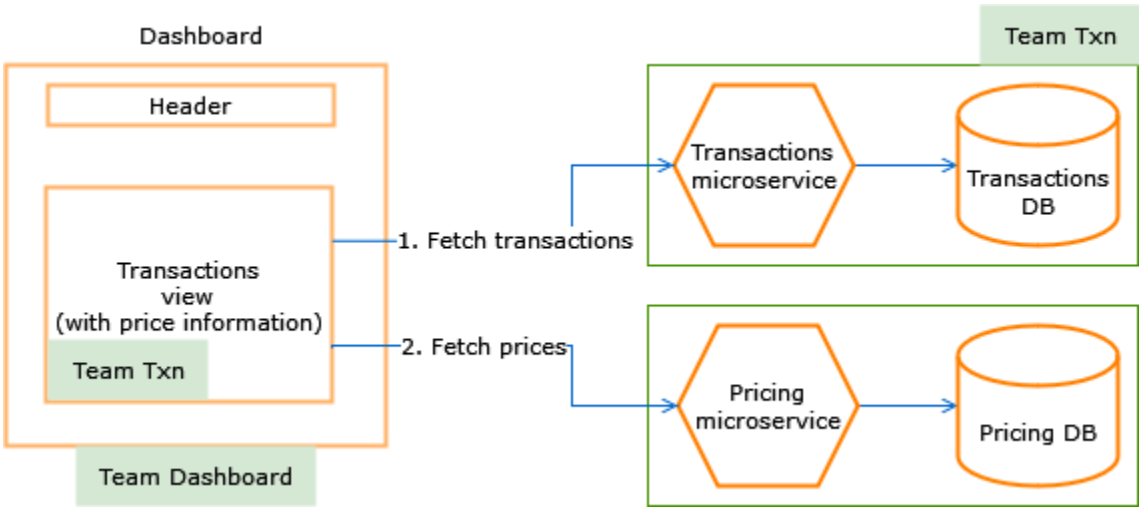
A diferencia de un servicio tradicional, una mejor amiga no sigue un modelo de dominio. En cambio, se trata de una capa de API para que la microinterfaz procese previamente los datos antes de que lleguen al cliente. Entre las áreas en las que esto resulta útil se incluyen las siguientes:

- Autorización para uso privado APIs
- Agregación de datos de diferentes fuentes
- Transformación de datos para reducir la carga de la red y facilitar el consumo de datos por parte del cliente

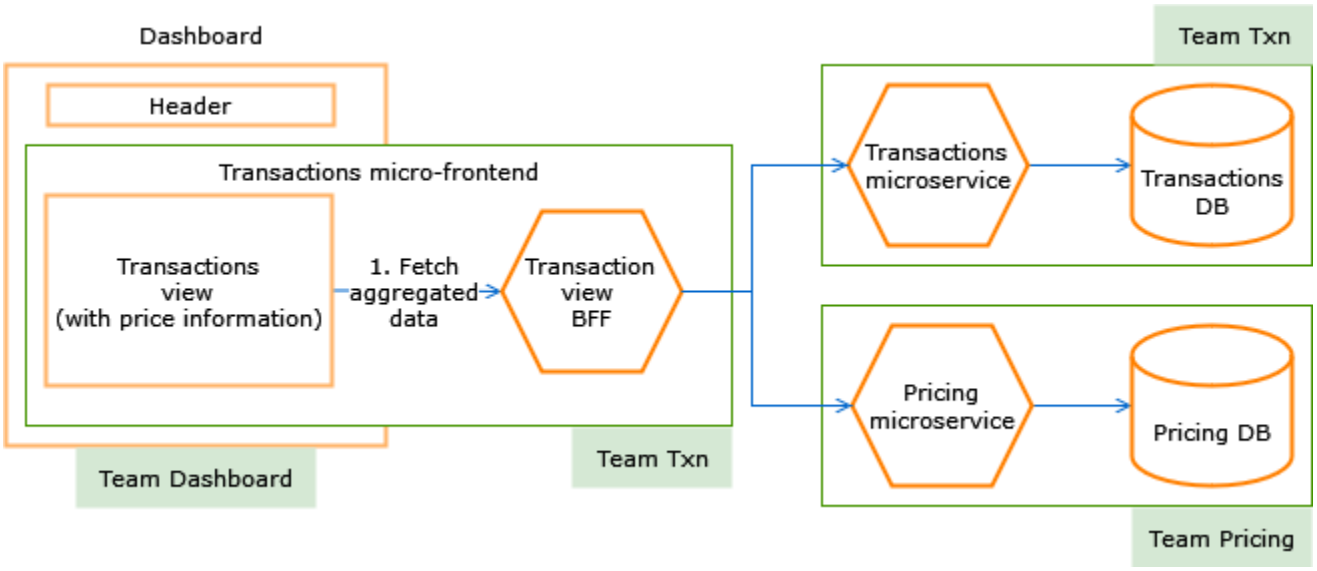
Por lo tanto, un BFF es propiedad de la microinterfaz, no del nivel de servicio del dominio. BFFs se puede implementar mediante lo siguiente:

- AWS AppSync GraphQL APIs
- Un conjunto de funciones de AWS Lambda
- Como contenedor que se ejecuta en Amazon ECS, Amazon EKS o AWS AppRunner

El siguiente diagrama muestra que, sin el patrón BFF, las microinterfaces deben conectarse a puntos de enlace individuales de la API de microservicios para obtener y agregar datos.



En cambio, con el patrón BFF del siguiente diagrama, las microinterfaces pueden comunicarse con su propio backend y obtener datos agregados.



Los equipos pueden desarrollar BFFs programas para distintos canales, como dispositivos móviles, sitios web o vistas específicas, con el objetivo de optimizar las interacciones internas reduciendo las conversaciones.

Estilismo y CSS

Las hojas de estilo en cascada (CSS) son un lenguaje para determinar de forma centralizada la presentación de un documento en lugar de codificar el formato del texto y los objetos. La función de cascada del lenguaje está diseñada para controlar las prioridades entre los estilos mediante el uso de la herencia. Cuando se trabaja con microinterfaces y se crea una estrategia para gestionar las dependencias, la función de cascada del lenguaje puede ser todo un desafío.

Por ejemplo, dos microinterfaces coexisten en la misma página y cada una define su propio estilo para el elemento HTML. `body` Si cada uno busca su propio archivo CSS y lo adjunta al DOM mediante una `style` etiqueta, el archivo CSS reemplaza al primero si ambos tienen una definición de elementos HTML, nombres de clases o elementos comunes. IDs Existen diferentes estrategias para abordar estos problemas, según la estrategia de dependencia que se elija para administrar los estilos.

En la actualidad, el enfoque más popular para equilibrar el rendimiento, la coherencia y la experiencia del desarrollador consiste en desarrollar y mantener un sistema de diseño.

Diseñar sistemas: un enfoque basado en compartir algo

Este enfoque utiliza un sistema para compartir estilos cuando es apropiado y, al mismo tiempo, admite divergencias ocasionales para equilibrar la coherencia, el rendimiento y la experiencia del desarrollador. Un sistema de diseño es un conjunto de componentes reutilizables, guiados por normas claras. El desarrollo del sistema de diseño suele estar impulsado por un equipo con las aportaciones y contribuciones de muchos equipos. En términos prácticos, un sistema de diseño es una forma de compartir elementos de bajo nivel que se pueden exportar como una JavaScript biblioteca. Los desarrolladores con microinterfaces pueden utilizar la biblioteca como una dependencia para crear interfaces sencillas componiendo los recursos disponibles previamente y como punto de partida para crear nuevas interfaces.

Considere el ejemplo de una microinterfaz que necesita un formulario. La experiencia típica de un desarrollador consiste en utilizar componentes prefabricados disponibles en el sistema de diseño para crear cuadros de texto, botones, listas desplegables y otros elementos de la interfaz de usuario. El desarrollador no necesita escribir ningún estilo para los componentes reales, solo para determinar el aspecto que tienen juntos. El sistema que se va a construir y lanzar puede utilizar webpack Module Federation o un enfoque similar para declarar el sistema de diseño como una dependencia externa, de modo que la lógica del formulario se empaquete sin incluir el sistema de diseño.

De este modo, varias microinterfaces pueden hacer lo mismo para resolver problemas comunes. Cuando los equipos desarrollan nuevos componentes que pueden compartirse entre múltiples microinterfaces, esos componentes se añaden al sistema de diseño una vez que alcanzan su madurez.

Una de las principales ventajas del enfoque del sistema de diseño es el alto nivel de coherencia. Si bien las microinterfaces pueden escribir estilos y, en ocasiones, anular los del sistema de diseño, es muy poco necesario. Los principales elementos de bajo nivel no cambian con frecuencia y ofrecen una funcionalidad básica que se puede ampliar de forma predeterminada. Otra ventaja es el rendimiento. Con una buena estrategia de creación y publicación, puede producir un mínimo de paquetes compartidos que estén instrumentados por el shell de la aplicación. Puede mejorar aún más si se cargan de forma asíncrona varios paquetes específicos de microinterfaz según demanda, con un consumo mínimo en términos de ancho de banda de red. Por último, pero no por ello menos importante, la experiencia de los desarrolladores es ideal, ya que las personas pueden centrarse en crear interfaces sofisticadas sin tener que reinventar la rueda (por ejemplo, escribir JavaScript y usar CSS cada vez que es necesario añadir un botón a una página).

La desventaja es que un sistema de diseño de cualquier tipo es una dependencia, por lo que debe mantenerse y, en ocasiones, actualizarse. Si varias microinterfaces requieren una nueva versión de una dependencia compartida, puede usar una de las siguientes opciones:

- Un mecanismo de organización que, de vez en cuando, puede recuperar varias versiones de esa dependencia compartida sin conflictos
- Una estrategia compartida para hacer que todos los dependientes usen una nueva versión

Por ejemplo, si todas las microinterfaces dependen de la versión 3.0 de un sistema de diseño y hay una nueva versión llamada 3.1 que se utiliza de forma compartida, puede implementar indicadores de características para que todas las microinterfaces migren con un riesgo mínimo. Para obtener más información, consulte la sección Indicadores de características. Otro posible inconveniente es que los sistemas de diseño suelen abordar algo más que el estilo. También incluyen JavaScript prácticas y herramientas. Estos aspectos requieren llegar a un consenso a través del debate y la colaboración.

Implementar un sistema de diseño es una buena inversión a largo plazo. Es un enfoque popular y cualquiera que trabaje en una arquitectura de frontend compleja debería tenerlo en cuenta. Por lo general, requiere que los ingenieros de front-end y los equipos de producto y diseño colaboren y definan mecanismos para interactuar entre sí. Es importante programar el tiempo para alcanzar el

estado deseado. También es importante contar con el patrocinio de los líderes para que las personas puedan construir algo confiable, bien mantenido y que rinda bien a largo plazo.

CSS totalmente encapsulado: un enfoque de no compartir nada

Cada microinterfaz utiliza convenciones y herramientas para superar la función de cascada del CSS. Un ejemplo es garantizar que el estilo de cada elemento esté siempre asociado a un nombre de clase en lugar de a su ID, y que los nombres de clase sean siempre únicos. De esta forma, todo depende de las microinterfaces individuales y se minimiza el riesgo de conflictos no deseados. Por lo general, el shell de la aplicación se encarga de cargar los estilos de las microinterfaces una vez que se han cargado en el DOM, aunque algunas herramientas agrupan los estilos mediante el uso de estos. JavaScript

La principal ventaja de no compartir nada es el menor riesgo de que se produzcan conflictos entre las microinterfaces. Otra ventaja es la experiencia del desarrollador. Cada microinterfaz no comparte nada con otras microinterfaces. Publicar y probar de forma aislada es más sencillo y rápido.

Una desventaja principal del enfoque de no compartir nada es la posible falta de coherencia. No existe ningún sistema para evaluar la coherencia. Aunque el objetivo es duplicar lo que se comparte, resulta difícil equilibrar la velocidad de publicación y la colaboración. Una mitigación habitual consiste en crear herramientas para medir la coherencia. Por ejemplo, puedes crear un sistema para realizar capturas de pantalla automatizadas de varias microinterfaces renderizadas en una página con un navegador sin interfaz gráfica. A continuación, puedes revisar manualmente las capturas de pantalla antes del lanzamiento. Sin embargo, eso requiere disciplina y gobierno. Para obtener más información, consulte la sección Equilibrar la autonomía con la alineación.

Según el caso de uso, otra posible desventaja es el rendimiento. Si todas las microinterfaces utilizan una gran cantidad de estilos, el cliente debe descargar una gran cantidad de código duplicado. Eso afectará negativamente a la experiencia del usuario.

Este enfoque de «no compartir nada» debería considerarse únicamente en el caso de arquitecturas de microinterfaces en las que participan solo unos pocos equipos, o en las microinterfaces que toleran una baja coherencia. También puede ser un paso inicial natural mientras una organización trabaja en un sistema de diseño.

CSS global compartido: un enfoque que permite compartirlo todo

Con este enfoque, todo el código relacionado con el estilo se almacena en un repositorio central donde los colaboradores escriben CSS para todas las microinterfaces trabajando en archivos CSS

o utilizando preprocesadores como Sass. Cuando se realizan cambios, un sistema de compilación crea un único paquete de CSS que se puede alojar en una CDN e incluir en cada microinterfaz mediante el shell de la aplicación. Los desarrolladores de microinterfaces pueden diseñar y crear sus aplicaciones ejecutando su código a través de una consola de aplicaciones alojada localmente.

Además de la evidente ventaja de reducir el riesgo de conflictos entre las microinterfaces, las ventajas de este enfoque son la coherencia y el rendimiento. Sin embargo, al separar los estilos del marcado y la lógica, los desarrolladores tienen más dificultades para entender cómo se utilizan los estilos, cómo pueden evolucionar y cómo pueden quedar obsoletos. Por ejemplo, puede resultar más rápido introducir un nombre de clase nuevo que aprender sobre la clase existente y las consecuencias de editar sus propiedades. Las desventajas de crear un nombre de clase nuevo son el aumento del tamaño del paquete, que afecta al rendimiento, y la posible introducción de incoherencias en la experiencia del usuario.

Si bien un CSS global compartido puede ser el punto de partida de una monolith-to-micro-frontends migración, rara vez resulta beneficioso para las arquitecturas microfrontend en las que participan más de uno o dos equipos que colaboran entre sí. Recomendamos invertir en un sistema de diseño lo antes posible e implementar un enfoque de no compartir nada mientras el sistema de diseño esté en desarrollo.

Organización y formas de trabajo

Como ocurre con todas las estrategias de arquitectura, las microinterfaces tienen implicaciones que van mucho más allá de la tecnología que una organización decida implementar. La decisión de crear aplicaciones microfrontales debe estar en consonancia con la empresa, el producto, la organización, las operaciones e incluso la cultura (por ejemplo, empoderando a los equipos y descentralizando la toma de decisiones). A cambio, este tipo de arquitectura de microinterfaz permite un desarrollo verdaderamente ágil e impulsado por el producto, ya que reduce considerablemente la sobrecarga de comunicación entre equipos que, de otro modo, serían independientes.

Desarrollo ágil

La idea del desarrollo ágil de software se ha vuelto tan universal en los últimos años que prácticamente todas las organizaciones afirman trabajar de forma ágil. Si bien una definición concluyente de ágil va más allá del alcance de esta estrategia, vale la pena revisar los elementos clave que son relevantes para el desarrollo de microinterfaces.

La base del paradigma ágil es el Manifiesto Ágil (2001), que postuló cuatro principios principales (por ejemplo, «Las personas y las interacciones en lugar de los procesos y las herramientas») y doce principios. Los marcos de procesos, como Scrum y Scaled Agile Framework (SAFe), surgieron en torno al Manifiesto Ágil y se han incorporado a las prácticas cotidianas. Sin embargo, la filosofía que los sustenta se malinterpreta o se ignora en gran medida.

En el contexto de la arquitectura de microinterfaz, es importante adoptar los siguientes principios ágiles:

- «Entregue software que funcione con frecuencia, de un par de semanas a un par de meses, con preferencia por un plazo más corto».

  Este principio hace hincapié en la importancia de trabajar en incrementos y entregar el software a la producción con la mayor regularidad y frecuencia posible. Desde una perspectiva tecnológica, esto se refiere a la integración y la entrega continuas (CI/CD). In CI/CDlas herramientas y los procesos de creación, prueba e implementación son partes integrales de cada proyecto de software). El principio también implica que la infraestructura de ejecución y las responsabilidades operativas deben ser responsabilidad del equipo. Esa propiedad es especialmente importante en los sistemas distribuidos, donde los subsistemas independientes pueden tener requisitos de infraestructura y operaciones significativamente diferentes.

- «Cree proyectos en torno a personas motivadas. Bríndeles el entorno y el apoyo que necesitan, y confíe en ellos para hacer su trabajo».

  «Las mejores arquitecturas, requisitos y diseños surgen de equipos que se autoorganizan».

  Ambos principios hacen hincapié en los beneficios de la propiedad, la independencia y end-to-end la responsabilidad. Una arquitectura de microfrontend tendrá éxito cuando (y solo cuando) los equipos sean realmente dueños de sus microfrontend. End-to-endLa responsabilidad, desde la concepción hasta el diseño y la implementación, hasta la entrega y el funcionamiento, garantiza que el equipo pueda realmente asumir la responsabilidad. Esta independencia es necesaria, tanto técnica como organizativamente, para que el equipo tenga autonomía en la dirección estratégica. No recomendamos utilizar una plataforma de microinterfaz en una organización centralizada que utilice un modelo de desarrollo en cascada.

Composición y tamaño del equipo

Para que un equipo de software ejerza la propiedad, debe gobernarse a sí mismo, incluyendo cómo y qué entrega el equipo, dentro de los límites impuestos por la organización.

Para ser eficaces, los equipos deben poder entregar software de forma independiente y tener la autoridad para decidir la mejor manera de entregarlo. No se puede considerar autónomo a un equipo que recibe los requisitos de características de un gerente de producto externo o los diseños de interfaz de usuario de un diseñador externo, sin participar en la planificación de estos elementos. Las funciones pueden infringir los contratos o la funcionalidad existentes. Estas infracciones requerirán más debates y negociaciones, con el riesgo de retrasar la entrega e introducir conflictos innecesarios entre los equipos.

Al mismo tiempo, los equipos no deberían ser demasiado grandes. Si bien un equipo más grande tiene más recursos y puede adaptarse a las ausencias individuales, la complejidad de la comunicación aumenta exponencialmente con cada nuevo miembro. No es posible establecer un tamaño máximo de equipo válido universalmente. La cantidad de personas necesarias para un proyecto depende de factores como la madurez del equipo, la complejidad técnica, el ritmo de innovación y la infraestructura. Por ejemplo, Amazon sigue la regla de las dos pizzas: un equipo que sea demasiado grande para alimentarse con dos pizzas debe dividirse en equipos más pequeños. Eso puede ser un desafío. La división debe producirse dentro de los límites naturales y debe dar a cada equipo autonomía y propiedad sobre su trabajo.

DevOps cultura

DevOps se refiere a una práctica de ingeniería de software en la que los pasos del ciclo de vida del desarrollo están estrechamente integrados desde una perspectiva organizativa y técnica. Contrariamente a la creencia popular, DevOps tiene mucho que ver con la cultura y la mentalidad, y muy poco con las funciones y las herramientas.

Tradicionalmente, una organización de software tenía equipos de especialistas, por ejemplo, para el diseño, la implementación, las pruebas, el despliegue y las operaciones. Cada vez que un equipo terminaba su trabajo, pasaba el proyecto al siguiente equipo. Sin embargo, la entrega del software a través de silos de equipos especializados provoca fricciones durante las entregas. Al mismo tiempo, cuando los especialistas se ven obligados a trabajar con un enfoque limitado, carecen de conocimientos en dominios vecinos y no tienen una visión sistémica del producto. Esos déficits pueden provocar una baja coherencia del producto de software.

Por ejemplo, cuando un arquitecto de software diseña una solución para ser implementada por alguien de un equipo diferente, puede pasar por alto aspectos inherentes a la implementación (como la falta de coincidencia de dependencias). Luego, los desarrolladores utilizan atajos (como un parche temporal) o back-and-forth se inicia una formalización entre el arquitecto y el equipo de desarrollo. Debido a la sobrecarga que supone gestionar estos procesos, el desarrollo ha dejado de ser ágil (en el sentido de flexible, adaptativo, incremental e informal).

Si bien el término DevOps se refiere principalmente a la cultura, implica las tecnologías y los procesos que hacen DevOps posible la práctica. DevOps está estrechamente relacionado con que CI/CD. When a developer has finished implementing an increment of software, they commit it to a version-control system such as Git. Traditionally, a build system would then build and integrate the software, and have it tested and released in a more or less unified and centralized process. With CI/CD la creación, la integración, las pruebas y el lanzamiento del software son inherentes y automatizados. Lo ideal es que el proceso forme parte del propio proyecto de software a través de archivos de configuración que se adapten específicamente al proyecto en cuestión.

Se automatizan tantos pasos como sea posible. Por ejemplo, deberían reducirse las prácticas de pruebas manuales, ya que casi todos los tipos de pruebas se pueden automatizar. Cuando el proyecto se configura de esa manera, las actualizaciones de un producto de software se pueden entregar varias veces al día con gran confianza. Otra tecnología compatible DevOps es la infraestructura como código (IaC).

Tradicionalmente, la configuración y el mantenimiento de la infraestructura de TI requerían el trabajo manual de instalar y mantener el hardware (configurar los cables y los servidores en un

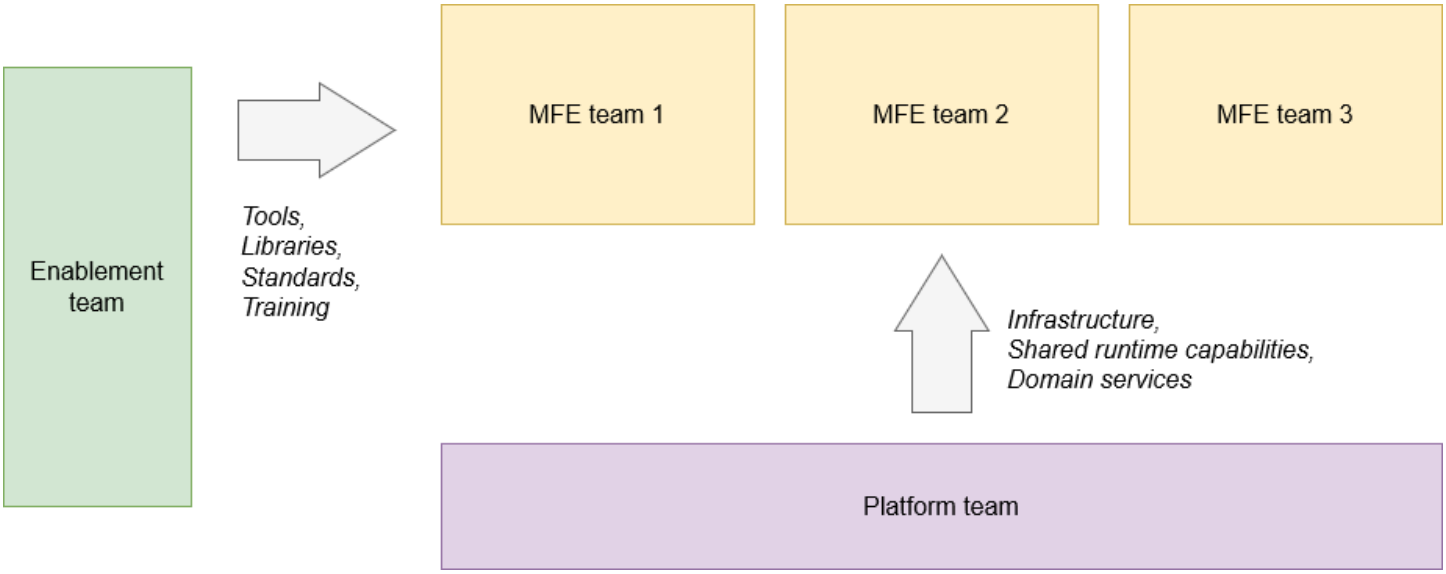
centro de datos) y el software operativo. Esto era necesario, pero tenía numerosos inconvenientes. La configuración llevaba mucho tiempo y era propensa a errores. A menudo, el hardware estaba sobreaprovisionado o insuficientemente, lo que generaba un gasto excesivo o una degradación del rendimiento. Al utilizar el iAC, puede describir los requisitos de infraestructura de un sistema de TI mediante un archivo de configuración desde el que se pueden implementar y actualizar automáticamente los servicios en la nube.

¿Qué tiene que ver todo esto con las microinterfaces? DevOps, CI/CD e iAc son complementos ideales para una arquitectura de microfrontend. Los beneficios de las microinterfaces se basan en procesos de entrega rápidos y sin fricciones. Una DevOps cultura solo puede prosperar en entornos en los que los equipos son responsables de los proyectos de software. end-to-end

Organizar el desarrollo de microinterfaces entre varios equipos

Al ampliar el desarrollo de la microinterfaz entre varios equipos multifuncionales, surgen dos problemas: en primer lugar, los equipos comienzan a desarrollar su propia interpretación del paradigma, a elegir el marco y la biblioteca y a crear sus propias bibliotecas de herramientas y auxiliares. En segundo lugar, los equipos totalmente autónomos deben asumir la responsabilidad de las capacidades genéricas, como la gestión de infraestructuras de bajo nivel. Por lo tanto, tiene sentido introducir dos equipos adicionales en una organización con microinterfaz integrada por varios equipos: un equipo de habilitación y un equipo de plataforma. Estos conceptos se adoptan ampliamente en las organizaciones de TI modernas con sistemas distribuidos y están bien documentados en las topologías de equipos.

En el siguiente diagrama se muestra al equipo de habilitación que proporciona herramientas, bibliotecas, estándares y pruebas a tres equipos de microinterfaz. El equipo de la plataforma proporciona infraestructura, capacidades de tiempo de ejecución compartidas y servicios de dominio a esos mismos tres equipos de microfrontend.



El equipo de la plataforma apoya a los equipos de microinterfaz al liberarlos del trabajo pesado indiferenciado. Este soporte puede incluir servicios de infraestructura, como tiempos de ejecución de contenedores, CI/CD canalizaciones, herramientas de colaboración y monitoreo. Sin embargo, la creación de un equipo de plataformas no debería conducir a una organización en la que el desarrollo esté separado de las operaciones. Es todo lo contrario: el equipo de la plataforma ofrece un producto de ingeniería y los equipos de microinterfaz son los propietarios y responsables de la ejecución de sus servicios en la plataforma.

El equipo de habilitación proporciona apoyo centrándose en la gobernanza y garantizando la coherencia entre los equipos de microinterfaz. (El equipo de la plataforma no debería participar en esto). El equipo de habilitación mantiene recursos compartidos, como una biblioteca de interfaz de usuario, y crea estándares como la elección del marco, los presupuestos de rendimiento y las convenciones de interoperabilidad. Al mismo tiempo, proporciona a los nuevos equipos o miembros del equipo formación sobre la aplicación de las normas y las herramientas definidas por la gobernanza.

Implementación

La clave de la autonomía de los equipos con microinterfaz es disponer de un proceso automatizado con un proceso de producción que sea independiente del de otros equipos microfrontend. Los equipos que siguen el principio de no compartir nada pueden implementar procesos independientes. Los equipos que comparten bibliotecas o dependen de un equipo de plataforma deben decidir cómo gestionar las dependencias en los procesos de implementación.

Por lo general, cada canalización hace lo siguiente:

- Crea activos de interfaz
- Despliega los activos en el alojamiento para su consumo
- Garantiza que los registros y las cachés se actualicen para que las nuevas versiones se puedan entregar a los clientes

Los pasos reales del proceso varían según la tecnología y el enfoque de composición de la página.

En el caso de la composición por parte del cliente, esto implica cargar los paquetes de aplicaciones en paquetes de alojamiento y distribuirlos para su consumo mediante el almacenamiento en caché en una CDN. Las aplicaciones que utilizan el almacenamiento en caché del navegador con los trabajadores del servicio también deberían implementar formas de actualizar las cachés de los trabajadores del servicio.

En el caso de la composición del lado del servidor, esto normalmente implica implementar la nueva versión del componente del servidor y actualizar el registro de la microinterfaz para que la nueva versión sea reconocible. Puede utilizar blue/green o canalizar los patrones de implementación para lanzar nuevas versiones de forma gradual.

Gobernanza

Por lo general, varias personas trabajan en microinterfaces, y cada una de ellas trabaja con diferentes restricciones para alcanzar objetivos empresariales comunes. Si bien la comunicación y la colaboración entre las personas son fundamentales para el éxito, la comunicación excesiva y la implementación de procesos demasiado complicados ralentizan el ciclo de desarrollo. Esto se traduce en una disminución de la moral y reduce el nivel de calidad.

Las empresas más exitosas que implementan microinterfaces mediante el uso de varios equipos crean mecanismos para equilibrar la autonomía con la alineación. Permiten a los responsables de la toma de decisiones tomar medidas a nivel local y escalar jerárquicamente solo cuando sea necesario. Los mecanismos incluyen los siguientes:

- Contratos de API
- Interactividad cruzada mediante eventos
- Equilibrar la autonomía con la alineación
- Banderas de características
- Detección de servicios

Contratos de API

Cada microinterfaz es un sistema capaz de encapsular opiniones, lógica y complejidad. Las preocupaciones transversales suelen incluir las siguientes:

- Sistemas de diseño: herramientas para el desarrollo UIs distribuidas en forma de bibliotecas
- Composición: la forma en que una microinterfaz necesita interactuar con el shell de una aplicación para renderizar y heredar su contexto
- Manejo lógico – Interacción con APIs para gestionar el estado persistente
- Interactividad con otras microinterfaces: escenarios como publicar y consumir eventos o navegar de una microinterfaz a otra

Para acelerar el consumo y la resolución de problemas, es habitual invertir en estandarizar la forma en que se declaran y documentan estas interfaces, incluidas las dependencias entre las microinterfaces. Los wikis creados por humanos son un buen comienzo. Un enfoque más escalable

consiste en almacenar esta información como metadatos estructurados en código. A continuación, puede centralizarla para su consumo mediante el uso de la automatización para realizar un seguimiento de los cambios históricos y proporcionar una búsqueda de texto completo.

Cuando las microinterfaces involucran a un gran número de equipos, se necesita una estrategia para coordinar los equipos. Compartir los contratos de API de forma unificada se convierte en algo imprescindible porque reduce la sobrecarga de comunicación y mejora la experiencia del desarrollador.

OpenAPI es un lenguaje de especificación para HTTP APIs que permite definir interfaces y contratos de API de forma unificada. Puede implementar REST APIs mediante OpenAPI en Amazon API Gateway. También puede utilizar una amplia variedad de marcos de código abierto que puede alojar en contenedores o máquinas virtuales. Una ventaja importante es que OpenAPI puede generar automáticamente documentación en un formato coherente, de modo que varios equipos pueden compartir conocimientos con una inversión inicial mínima.

Cuando varios equipos trabajan en microinterfaces, suelen formar grupos. En estos grupos, las personas pueden conocerse y aprender unas de otras y, al mismo tiempo, pensar en un panorama más amplio y contribuir a él. Estas iniciativas suelen definir y documentar los límites de propiedad, debatir cuestiones intersectoriales e identificar desde el principio cualquier duplicación de esfuerzos para resolver problemas comunes.

Interactividad cruzada mediante eventos

En algunos escenarios, es posible que sea necesario que varias microinterfaces interactúen entre sí para reaccionar ante los cambios de estado o las acciones de los usuarios. Por ejemplo, varias microinterfaces de la página pueden incluir menús plegables. Aparece un menú cuando el usuario selecciona un botón. El menú se oculta cuando el usuario hace clic en cualquier otro lugar, incluido otro menú que se muestra en una microinterfaz diferente.

Técnicamente, una biblioteca estatal compartida, como Redux, puede ser utilizada por múltiples microinterfaces y estar coordinada por una interfaz. Sin embargo, esto crea un acoplamiento significativo entre las aplicaciones, lo que hace que el código sea más difícil de probar y podría ralentizar el rendimiento durante el renderizado.

Un enfoque común y eficaz consiste en desarrollar un bus de eventos distribuido como una biblioteca, orquestado por el shell de la aplicación y utilizado por múltiples microinterfaces. De esta forma, cada microinterfaz publica y escucha determinados eventos de forma asíncrona, basando

su comportamiento únicamente en su propio estado interno. De este modo, varios equipos pueden mantener una página wiki compartida que describa los eventos y documente los comportamientos acordados por los diseñadores de experiencia de usuario.

En una implementación del ejemplo del bus de eventos, un componente desplegable utiliza el bus compartido para publicar un evento llamado `drop-down-open-menu` con una carga útil de. `{"id": "homepage-aboutus-button"}` El componente añade un detector al `drop-down-open-menu` evento para garantizar que, si se activa un evento con una nueva ID, el componente desplegable se renderiza para ocultar su sección plegable. De esta forma, la microinterfaz puede reaccionar a los cambios de forma asíncrona con un mayor rendimiento y una mejor encapsulación, lo que facilita que varios equipos diseñen y prueben los comportamientos.

Recomendamos utilizar un estándar APIs implementado de forma nativa en los navegadores modernos para mejorar la simplicidad y la facilidad de mantenimiento. La referencia de eventos de MDN proporciona información sobre el uso de eventos con aplicaciones renderizadas del lado del cliente.

Equilibrar la autonomía con la alineación

Las arquitecturas microfrontend están fuertemente sesgadas hacia la autonomía de los equipos. Sin embargo, es importante distinguir entre las áreas que pueden respaldar la flexibilidad y los diversos enfoques para resolver problemas, y las áreas en las que la estandarización es necesaria para lograr la alineación. Los principales líderes y arquitectos deben identificar estas áreas desde el principio y priorizar las inversiones para equilibrar la seguridad, el rendimiento, la excelencia operativa y la confiabilidad de las microinterfaces. Encontrar este equilibrio implica lo siguiente: crear, probar, lanzar y registrar las microinterfaces, monitorizar y emitir alertas.

Crear microinterfaces

Lo ideal es que todos los equipos estén estrechamente alineados para maximizar los beneficios en términos de rendimiento de los usuarios finales. En la práctica, esto puede resultar difícil y puede requerir más esfuerzo. Recomendamos comenzar con algunas pautas escritas a las que puedan contribuir varios equipos mediante un debate abierto y transparente. Luego, los equipos pueden adoptar gradualmente el patrón de software Cookiecutter, que permite crear herramientas que proporcionen una forma unificada de estructurar un proyecto.

Con este enfoque, puedes incorporar opiniones y restricciones. La desventaja es que estas herramientas requieren una inversión significativa para su creación y mantenimiento, y para

garantizar que los bloqueos se aborden rápidamente sin afectar a la productividad de los desarrolladores.

End-to-end pruebas de microinterfaces

Las pruebas unitarias se pueden dejar en manos de los propietarios. Recomendamos implementar una estrategia desde el principio para realizar pruebas cruzadas de microinterfaces que se ejecutan en una consola única. La estrategia incluye la capacidad de probar las aplicaciones antes y después de una versión de producción. Recomendamos desarrollar procesos y documentación para que el personal técnico y no técnico pueda probar las funcionalidades críticas de forma manual.

Es importante asegurarse de que los cambios no degraden la experiencia del cliente, funcional o no funcional. Una estrategia ideal es invertir gradualmente en pruebas automatizadas, tanto para las funciones clave como para las características de la arquitectura, como la seguridad y el rendimiento.

Lanzamiento de microinterfaces

Cada equipo puede tener su propia forma de implementar su código, generar opiniones y tener su propia infraestructura. El coste de la complejidad del mantenimiento de estos sistemas suele ser un elemento disuasorio. Por el contrario, recomendamos invertir desde el principio para implementar una estrategia compartida que pueda aplicarse mediante herramientas compartidas.

Desarrolle plantillas con la plataforma de CI/CD que prefiera. Luego, los equipos pueden usar las plantillas previamente aprobadas y la infraestructura compartida para publicar los cambios en la producción. Puede empezar a invertir en este trabajo de desarrollo cuanto antes, ya que estos sistemas rara vez necesitan actualizaciones importantes tras un período inicial de pruebas y consolidación.

Registro y supervisión

Cada equipo puede tener diferentes métricas empresariales y de sistemas de las que quiera hacer un seguimiento con fines operativos o analíticos. Aquí también se puede aplicar el patrón del software Cookiecutter. La distribución de los eventos puede resumirse y ponerse a disposición como una biblioteca que pueden utilizar varias microinterfaces. Para equilibrar la flexibilidad y la autonomía, desarrolle herramientas para registrar métricas personalizadas y crear paneles o informes personalizados. Los informes promueven una estrecha colaboración con los propietarios de los productos y reducen el ciclo de comentarios de los clientes finales.

Al estandarizar la entrega, varios equipos pueden colaborar para realizar un seguimiento de las métricas. Por ejemplo, un sitio web de comercio electrónico puede realizar un seguimiento del

recorrido del usuario desde la microinterfaz «Detalles del producto» hasta la microinterfaz «Carrito» y luego la microinterfaz «Compra» para medir la participación, la pérdida de clientes y los problemas. Si cada microinterfaz registra los eventos mediante una única biblioteca, puedes consumir estos datos en su totalidad, explorarlos de forma integral e identificar tendencias interesantes.

Alertas

Al igual que el registro y la supervisión, las alertas se benefician de la estandarización y ofrecen cierto grado de flexibilidad. Los distintos equipos pueden reaccionar de forma diferente a las alertas funcionales y no funcionales. Sin embargo, si todos los equipos tienen una forma consolidada de iniciar las alertas en función de las métricas recopiladas y analizadas en una plataforma compartida, la empresa puede identificar los problemas entre los equipos. Esta capacidad resulta útil durante los eventos de gestión de incidentes. Por ejemplo, las alertas se pueden iniciar de la siguiente manera:

- Número elevado de excepciones del JavaScript lado del cliente en una versión concreta del navegador
- El tiempo necesario para renderizar se ha degradado significativamente por encima de un umbral determinado
- Número elevado de códigos de estado de 5xx cuando se consume una API determinada

Según la madurez de su sistema, puede equilibrar sus esfuerzos en diferentes partes de su infraestructura, como se muestra en la siguiente tabla.

Adopción	Investigación y desarrollo	Ascenso	Madurez
Crea microinterfaces.	Experimenta, documenta y comparte lo aprendido.	Invierta en herramientas para crear nuevas microinterfaces. Evangeliza la adopción.	Consolide las herramientas para andamios. Impulse la adopción.
Pruebe las microinterfaces de	Implemente mecanismos para probar manualmente todas las	Invierta en herramientas para realizar pruebas automatizadas de seguridad y rendimiento. Investigue los indicadores de características y	Consolide las herramientas para la detección de servicios, las pruebas en producción y end-to-end las pruebas automatizadas.

Adopción	Investigación y desarrollo	Ascenso	Madurez
principio a fin.	microinterfaces relacionadas.	el descubrimiento de servicios.	
Libere microinterfaces.	Invierta en una infraestructura de CI/CD compartida y en versiones automatizadas para varios entornos. Evangelice la adopción.	Consolide las herramientas para la infraestructura de CI/CD Implemente mecanismos de reversión manual. Impulse la adopción.	Cree mecanismos para iniciar reversiones automatizadas a partir de las métricas y alertas del sistema y de la empresa.
Observe el rendimiento de la microinterfaz.	Invierta en una biblioteca e infraestructura de monitoreo compartidas para registrar de manera uniforme los eventos empresariales y del sistema.	Consolide las herramientas para la supervisión y las alertas. Implemente paneles de control entre equipos para supervisar el estado general y mejorar la gestión de incidentes.	Estandarice los esquemas de registro. Optimice los costos. Implemente alertas basadas en métricas empresariales complejas.

Marcas de características

Los indicadores de características se pueden implementar en las microinterfaces para facilitar la coordinación de las pruebas y el lanzamiento de funciones en varios entornos. La técnica de los marcadores de características consiste en centralizar las decisiones en una tienda basada en valores booleanos y, en función de ello, impulsar el comportamiento. Suele utilizarse para propagar de forma silenciosa cambios que pueden mantenerse ocultos hasta un momento concreto. Además, permite desbloquear nuevas versiones de nuevas funciones que, de otro modo, quedarían bloqueadas, lo que reduce la velocidad del equipo.

Pensemos en el ejemplo de equipos que trabajan en una función de microinterfaz que se lanzará en una fecha específica. La función está lista, pero debe publicarse junto con un cambio en otra microinterfaz que se publique de forma independiente. Bloquear el lanzamiento de ambas microinterfaces se consideraría contrario al patrón y aumentaría el riesgo una vez implementadas.

En su lugar, los equipos pueden crear un marcador de características booleano en una base de datos que ambos utilicen durante el procesamiento (por ejemplo, mediante una llamada HTTP a una API de Feature Flags compartida). Los equipos pueden incluso publicar el cambio en un entorno de prueba en el que el valor booleano esté establecido para verificar los requisitos funcionales y no funcionales de todos los proyectos antes de lanzarlos `True` a producción.

Otro ejemplo del uso de un indicador de función es la implementación de un mecanismo para anular el valor de un indicador estableciendo un valor específico a través del `QueryString` parámetro o almacenando una cadena de prueba determinada en una cookie. Los propietarios de los productos pueden iterar las funciones sin bloquear el lanzamiento de otras funciones ni corregir errores hasta la fecha de lanzamiento. En una fecha determinada, si se cambia el valor del indicador en la base de datos, el cambio se hace visible de forma instantánea en la fase de producción, sin necesidad de realizar publicaciones coordinadas entre equipos. Tras el lanzamiento de una función, los equipos de desarrollo limpian el código para eliminar el comportamiento anterior.

Otros casos de uso incluyen el lanzamiento de un sistema de indicadores de características basado en el contexto. Por ejemplo, si un solo sitio web atiende a clientes en varios idiomas, es posible que una función solo esté disponible para los visitantes de un país en particular. El sistema de indicadores de características puede depender de que el consumidor envíe el contexto del país (por ejemplo, utilizando el encabezado `Accept-Language` HTTP), y puede haber un comportamiento diferente en función de ese contexto.

Si bien los marcadores de características son una herramienta poderosa para facilitar la colaboración entre los desarrolladores y los propietarios de los productos, dependen de la diligencia de las personas para evitar una degradación significativa de la base de código. Mantener los indicadores activos en varias funciones puede aumentar la complejidad a la hora de solucionar problemas, aumentar el tamaño de los JavaScript paquetes y, en última instancia, acumular deudas técnicas. Entre las actividades de mitigación más comunes se incluyen las siguientes:

- Se realizan pruebas unitarias de cada función detrás de una bandera para reducir la probabilidad de que se produzcan errores, lo que puede provocar ciclos de retroalimentación más largos en los procesos automatizados de CI/CD que ejecutan las pruebas

- La creación de herramientas para medir el tamaño de los paquetes aumenta durante los cambios de código, lo que se puede mitigar durante las revisiones del código

AWS ofrece una gama de soluciones para optimizar las pruebas A/B in situ mediante el uso de Amazon CloudFront Functions o Lambda @Edge. Estos enfoques ayudan a reducir la complejidad de integrar una solución o el producto SaaS existente que está utilizando para hacer valer sus suposiciones. Para obtener más información, consulte las pruebas A/B.

Detección de servicios

El patrón de descubrimiento de interfaces mejora la experiencia de desarrollo a la hora de desarrollar, probar y entregar microinterfaces. El patrón utiliza una configuración que se puede compartir y que describe el punto de entrada de las microinterfaces. La configuración que se puede compartir también incluye metadatos adicionales que se utilizan para realizar implementaciones seguras en cada entorno mediante el uso de versiones de Canary.

El desarrollo de interfaces modernas implica el uso de una amplia variedad de herramientas y bibliotecas para respaldar la modularidad durante el desarrollo. Tradicionalmente, este proceso consistía en agrupar el código en archivos individuales que podían alojarse en una CDN con el objetivo de reducir al mínimo las llamadas de red durante el tiempo de ejecución, incluida la carga inicial (cuando una aplicación se abre en un navegador) y el uso (cuando un cliente realiza acciones como elegir botones o insertar información).

Dividir paquetes

Las arquitecturas microfrontend resuelven los problemas de rendimiento causados por los paquetes muy grandes que se generan al agrupar individualmente un gran conjunto de funcionalidades. Por ejemplo, un sitio web de comercio electrónico muy grande se puede agrupar en un archivo de 6 MB. JavaScript A pesar de la compresión, el tamaño de ese archivo puede afectar negativamente a la experiencia del usuario al cargar la aplicación y descargar el archivo desde una CDN optimizada para aplicaciones avanzadas.

Si divides la aplicación en microinterfaces para la página de inicio, los detalles del producto y el carrito de compras, puedes utilizar un mecanismo de agrupamiento para crear tres paquetes individuales de 2 MB. Este cambio podría mejorar el rendimiento en la primera carga en un 300 por ciento cuando los usuarios usen la página de inicio. Los paquetes de microinterfaces de productos o carritos se cargan de forma asíncrona solo si el usuario visita la página del producto en busca de un artículo y decide comprarlo.

Hay muchos marcos y bibliotecas disponibles basados en este enfoque, y esto supone ventajas tanto para los clientes como para los desarrolladores. Para identificar los límites empresariales que pueden resultar en la disociación de las dependencias en el código, puede asignar diferentes funciones empresariales a varios equipos. La propiedad distribuida aporta independencia y agilidad.

Al dividir los paquetes de compilación, puede usar una configuración para mapear las microinterfaces e impulsar la organización de la navegación durante la carga inicial y posterior a la carga. Luego, la configuración se puede consumir durante el tiempo de ejecución en lugar de durante el tiempo de compilación. Por ejemplo, el código del front-end del lado del cliente o el código del backend del lado del servidor pueden realizar una llamada de red inicial a una API para obtener dinámicamente la lista de microinterfaces. También obtiene los metadatos necesarios para la composición y la integración. Puede configurar las estrategias de conmutación por error y el almacenamiento en caché para garantizar la fiabilidad y el rendimiento. El mapeo de las microinterfaces permite que las implementaciones individuales de las microinterfaces sean detectadas por las microinterfaces previamente implementadas y orquestadas por una aplicación shell.

Lanzamientos de Canary

Una versión canaria es un patrón popular y bien establecido para implementar microservicios. Las versiones de Canary agrupan a los usuarios objetivo de una versión en varios grupos, y las versiones cambian de forma gradual, en lugar de sustituirlas inmediatamente (lo que también se conoce como despliegue azul/verde). Un ejemplo de estrategia de lanzamiento rápido consiste en implementar un nuevo cambio para el 10 por ciento de los usuarios objetivo y añadir un 10 por ciento cada minuto, con una duración total de 10 minutos para llegar al 100 por ciento.

El objetivo de una versión preliminar es obtener comentarios tempranos sobre los cambios y monitorear el sistema para reducir el impacto de cualquier problema. Cuando existe la automatización, un sistema interno puede supervisar las métricas de la empresa o del sistema y detener la implementación o iniciar una reversión.

Por ejemplo, un cambio puede provocar un error que, en los primeros minutos de una publicación, provoque una pérdida de ingresos o una degradación del rendimiento. La supervisión automatizada puede activar una alarma. Con el patrón de detección de servicios, esa alarma puede detener la implementación y revertirla inmediatamente, lo que afecta solo al 20 por ciento de los usuarios en lugar del 100 por ciento. La empresa se beneficia de la reducción del alcance del problema.

Para ver un ejemplo de arquitectura que usa DynamoDB como almacenamiento para implementar una API de administración de REST, consulte la solución Frontend Service Discovery on AWS en. GitHub Utilice la AWS CloudFormation plantilla para integrar la arquitectura en sus propias

canalizaciones de CI/CD. La solución incluye una API REST Consumer para integrar la solución con sus aplicaciones frontend.

¿Necesitas un equipo de plataformas?

Algunas empresas tienen un equipo que se encarga de gestionar y mantener el código, la infraestructura y los procesos que adoptan otros equipos para trabajar en las microinterfaces. Entre las responsabilidades comunes se incluyen las siguientes:

- Cree y mantenga una canalización de CI/CD que pueda usarse con repositorios que contengan microinterfaces. Cree y pruebe los cambios de código y libérelos en varios entornos.
- Cree y mantenga herramientas relacionadas con la observabilidad, como paneles compartidos, mecanismos de alerta y sistemas para reaccionar ante los problemas.
- Cree y mantenga bibliotecas compartidas para la gestión de eventos, el consumo de servicios compartidos y las dependencias de terceros.
- Cree y mantenga herramientas que supervisen continuamente las características no funcionales, como el rendimiento, la seguridad y la confiabilidad del sistema.
- Cree y mantenga sistemas de diseño.
- Cree, mantenga y respalde la carcasa de la aplicación para el sistema de microinterfaz.

Según la escala del proyecto, puede gestionar estas responsabilidades mediante uno de los siguientes enfoques:

- Cree un equipo de plataforma dedicado cuya única responsabilidad sea trabajar en herramientas compartidas.
- Crea un grupo compuesto por miembros de varios equipos. Los miembros del grupo dividen su tiempo entre trabajar en microinterfaces y trabajar en herramientas compartidas. Esto también se conoce como equipo tigre.

Si bien el enfoque de equipo tigre es una forma eficaz de centrarse en el cliente, un equipo tigre suele convertirse en un equipo de plataforma si el proyecto gana terreno y responsabilidades. Tanto en el caso de los equipos de plataformas como en los equipos de tipo tigre, las empresas más exitosas que trabajan con microinterfaces forman estos equipos para que puedan contribuir varias personas con distintos antecedentes y habilidades. Los miembros del equipo pueden incluir ingenieros de backend, ingenieros de front-end, diseñadores de experiencia de usuario (UX) y gerentes técnicos de productos. Esta diversidad impulsa a las personas a participar continuamente en debates saludables y a diseñar pensando en la simplicidad.

Siguientes pasos

Esta guía abordó los patrones arquitectónicos y organizativos, las ventajas y desventajas de las decisiones clave y las cuestiones de gobernanza relacionadas con las microinterfaces. Las tablas resumen las ventajas y desventajas de las prácticas analizadas en este documento en términos de las siguientes dimensiones:

- Autonomía: la capacidad de cada equipo de microinterfaz para desarrollar de forma independiente su implementación y lanzamiento para los usuarios finales.
- Coherencia: la experiencia general de la aplicación, en la que cada microinterfaz se comporta según lo esperado. La alta consistencia significa que las microinterfaces son coherentes con el resto de la aplicación y no perjudican la experiencia de usuario de la aplicación en general.
- Complejidad: la cantidad de infraestructura, código y esfuerzo necesarios para implementar y probar las microinterfaces, la aplicación en general y los controles de gobierno.

Práctica	Autonomía	Coherencia	Complejidad
Construi con microinte rfaces en lugar de aplicacio nes monolític as	Alta	Medio	Alta

Prácticas de código compartido	Autonomía	Coherencia	Complejidad
No comparta nada	Alta	Baja	Baja
Compart preocupa iones transvers ales	Medio	Alta	Medio
Compart la lógica empresa al	Baja	Alta	Medio
Compart a través de las bibliotec as en el momento de la compilac ón	Medio	Alta	Baja
Compart en tiempo	Alta	Alta	Alta

Prácticas de código compartido	Autonomía	Coherencia	Complejidad
de ejecución			

Prácticas de descubrimiento mediante microinterfaz	Autonomía	Coherencia	Complejidad
Configuración durante la creación de la aplicación	Baja	Alta	Baja
Descubrimiento del lado del servidor	Alta	Alta	Medio
Descubrimiento del lado	Alta	Alta	Medio

Prácticas de descubrimiento mediante microinterfaz	Autonomía	Coherencia	Complejidad
del cliente (en tiempo de ejecución)			

Vea las prácticas de composición	Autonomía	Coherencia	Complejidad
Composición del servidor	Alta	Medio	Alta
Composición de bordes y lados	Medio	Medio	Alta
Composición del cliente	Alta	Medio	Medio

Para obtener más información sobre los conceptos presentados en esta guía, consulte la sección de Recursos.

Recursos

- Las microinterfaces en su contexto
- Diseño impulsado por el dominio
- Imágenes EDA
- Descubrimiento de frontend
- Detección de servicios frontend en AWS
- Manifiesto ágil
- Referencia del evento MDN
- OpenAPI

Colaboradores

Las siguientes personas contribuyeron a esta guía.

- Matteo Figus, arquitecto principal de soluciones, AWS
- Alexander Guensche, arquitecto sénior de soluciones, AWS
- Harun Hasdal, arquitecto sénior de soluciones, AWS
- Luca Mezzalira, principal arquitecto de soluciones especializado en comercialización de Serverless UK, AWS

Historial del documento

En la siguiente tabla, se describen cambios significativos de esta guía. Si quiere recibir notificaciones de futuras actualizaciones, puede suscribirse a las notificaciones RSS.

Cambio	Descripción	Fecha
Publicación inicial	—	12 de julio de 2024

AWS Glosario de orientación prescriptiva

Los siguientes son términos de uso común en las estrategias, guías y patrones proporcionados por la Guía AWS prescriptiva. Para sugerir entradas, utilice el enlace Enviar comentarios al final del glosario.

Números

Las 7 R

Siete estrategias de migración comunes para trasladar aplicaciones a la nube. Estas estrategias se basan en las 5 R que Gartner identificó en 2011 y consisten en lo siguiente:

- Refactorizar/rediseñar: traslade una aplicación y modifique su arquitectura mediante el máximo aprovechamiento de las características nativas en la nube para mejorar la agilidad, el rendimiento y la escalabilidad. Por lo general, esto implica trasladar el sistema operativo y la base de datos. Ejemplo: Migrar la base de datos de Oracle en las instalaciones a Amazon Aurora PostgreSQL-Compatible Edition.
- Redefinir la plataforma (transportar y redefinir): traslade una aplicación a la nube e introduzca algún nivel de optimización para aprovechar las capacidades de la nube. Ejemplo: Migrar la base de datos Oracle en las instalaciones a Amazon Relational Database Service (Amazon RDS) para Oracle en la nube de Nube de AWS.
- Recomprar (readquirir): cambie a un producto diferente, lo cual se suele llevar a cabo al pasar de una licencia tradicional a un modelo SaaS. Ejemplo: Migrar el sistema de administración de las relaciones con los clientes (CRM) a Salesforce.com.
- Volver a alojar (migrar mediante lift-and-shift): traslade una aplicación a la nube sin realizar cambios para aprovechar las capacidades de la nube. Ejemplo: Migrar la base de datos de Oracle en las instalaciones a Oracle en una instancia de EC2 en la Nube de AWS.
- Reubicar: (migrar el hipervisor mediante lift and shift): traslade la infraestructura a la nube sin comprar equipo nuevo, reescribir aplicaciones o modificar las operaciones actuales. Los servidores se migran de una plataforma en las instalaciones a un servicio en la nube para la misma plataforma. Ejemplo: migrar una Microsoft Hyper-V aplicación a AWS.
- Retener (revisitar): conserve las aplicaciones en el entorno de origen. Estas pueden incluir las aplicaciones que requieren una refactorización importante, que desee posponer para más adelante, y las aplicaciones heredadas que desee retener, ya que no hay ninguna justificación empresarial para migrarlas.

- Retirar: retire o elimine las aplicaciones que ya no sean necesarias en un entorno de origen.

A

ABAC

Consulte control de acceso basado en atributos.

servicios abstractos

Consulte servicios administrados.

ACID

Consulte atomicidad, consistencia, aislamiento, durabilidad.

migración activa-activa

Método de migración de bases de datos en el que las bases de datos de origen y destino se mantienen sincronizadas (mediante una herramienta de replicación bidireccional o mediante operaciones de escritura doble) y ambas bases de datos gestionan las transacciones de las aplicaciones conectadas durante la migración. Este método permite la migración en lotes pequeños y controlados, en lugar de requerir una transición única. Es más flexible, pero requiere más trabajo que una migración activa-pasiva.

migración activa-pasiva

Método de migración de bases de datos en el que las bases de datos de origen y destino se mantienen sincronizadas, pero solo la de origen gestiona las transacciones de las aplicaciones conectadas, mientras los datos se replican en la de destino. La base de datos de destino no acepta ninguna transacción durante la migración.

función de agregación

Función SQL que actúa en un grupo de filas y calcula un único valor de devolución para el grupo. Entre los ejemplos de funciones de agregación se incluyen `SUM` y `MAX`.

IA

Consulte inteligencia artificial.

AIOps

Consulte operaciones de inteligencia artificial

anonimización

El proceso de eliminar permanentemente la información personal de un conjunto de datos. La anonimización puede ayudar a proteger la privacidad personal. Los datos anonimizados ya no se consideran datos personales.

antipatrones

Una solución que se utiliza con frecuencia para un problema recurrente en el que la solución es contraproducente, ineficaz o menos eficaz que una alternativa.

control de aplicaciones

Enfoque de seguridad que permite usar de manera exclusiva aplicaciones aprobadas para ayudar a proteger un sistema contra el malware.

cartera de aplicaciones

Recopilación de información detallada sobre cada aplicación que utiliza una organización, incluido el costo de creación y mantenimiento de la aplicación y su valor empresarial. Esta información es clave para el proceso de detección y análisis de la cartera y ayuda a identificar y priorizar las aplicaciones que se van a migrar, modernizar y optimizar.

inteligencia artificial (IA)

El campo de la informática que se dedica al uso de tecnologías informáticas para realizar funciones cognitivas que suelen estar asociadas a los seres humanos, como el aprendizaje, la resolución de problemas y el reconocimiento de patrones. Para más información, consulte ¿Qué es la inteligencia artificial?

operaciones de inteligencia artificial (AIOps)

El proceso de utilizar técnicas de machine learning para resolver problemas operativos, reducir los incidentes operativos y la intervención humana, y mejorar la calidad del servicio. Para obtener más información sobre cómo AIOps se utiliza en la estrategia de AWS migración, consulte la guía de integración de operaciones.

cifrado asimétrico

Algoritmo de cifrado que utiliza un par de claves, una clave pública para el cifrado y una clave privada para el descifrado. Puede compartir la clave pública porque no se utiliza para el descifrado, pero el acceso a la clave privada debe estar sumamente restringido.

atomicidad, consistencia, aislamiento, durabilidad (ACID)

Conjunto de propiedades de software que garantizan la validez de los datos y la fiabilidad operativa de una base de datos, incluso en caso de errores, cortes de energía u otros problemas.

control de acceso basado en atributos (ABAC)

La práctica de crear permisos detallados basados en los atributos del usuario, como el departamento, el puesto de trabajo y el nombre del equipo. Para obtener más información, consulte ABAC AWS en la documentación AWS Identity and Access Management (IAM).

origen de datos fidedigno

Ubicación en la que se almacena la versión principal de los datos, que se considera la fuente de información más fiable. Puede copiar los datos del origen de datos autorizado a otras ubicaciones con el fin de procesarlos o modificarlos, por ejemplo, anonimizarlos, redactarlos o seudonimizarlos.

Zona de disponibilidad

Una ubicación distinta dentro de una Región de AWS que está aislada de los fallos en otras zonas de disponibilidad y que proporciona una conectividad de red económica y de baja latencia a otras zonas de disponibilidad de la misma región.

AWS Marco de adopción de la nube (AWS CAF)

Un marco de directrices y mejores prácticas AWS para ayudar a las organizaciones a desarrollar un plan eficiente y eficaz para migrar con éxito a la nube. AWS CAF organiza la orientación en seis áreas de enfoque denominadas perspectivas: negocios, personas, gobierno, plataforma, seguridad y operaciones. Las perspectivas empresariales, humanas y de gobernanza se centran en las habilidades y los procesos empresariales; las perspectivas de plataforma, seguridad y operaciones se centran en las habilidades y los procesos técnicos. Por ejemplo, la perspectiva humana se dirige a las partes interesadas que se ocupan de los Recursos Humanos (RR. HH.), las funciones del personal y la administración de las personas. Desde esta perspectiva, AWS CAF proporciona orientación para el desarrollo, la formación y la comunicación de las personas a fin de preparar a la organización para una adopción exitosa de la nube. Para obtener más información, consulte la Página web de AWS CAF y el Documento técnico de AWS CAF.

AWS Marco de calificación de la carga de trabajo (AWS WQF)

Herramienta que evalúa las cargas de trabajo de migración de bases de datos, recomienda estrategias de migración y proporciona estimaciones de trabajo. AWS WQF se incluye con AWS

Schema Conversion Tool ().AWS SCT Analiza los esquemas de bases de datos y los objetos de código, el código de las aplicaciones, las dependencias y las características de rendimiento y proporciona informes de evaluación.

B

bot malicioso

Bot destinado a causar interrupciones o daños a personas u organizaciones.

BCP

Consulte planificación de la continuidad del negocio.

gráfico de comportamiento

Una vista unificada e interactiva del comportamiento de los recursos y de las interacciones a lo largo del tiempo. Puede utilizar un gráfico de comportamiento con Amazon Detective para examinar los intentos de inicio de sesión fallidos, las llamadas sospechosas a la API y acciones similares. Para obtener más información, consulte Datos en un gráfico de comportamiento en la documentación de Detective.

sistema big-endian

Un sistema que almacena primero el byte más significativo. Consulte también endianidad.

clasificación binaria

Un proceso que predice un resultado binario (una de las dos clases posibles). Por ejemplo, es posible que su modelo de ML necesite predecir problemas como “¿Este correo electrónico es spam o no es spam?” o “¿Este producto es un libro o un automóvil?”.

filtro de floración

Estructura de datos probabilística y eficiente en términos de memoria que se utiliza para comprobar si un elemento es miembro de un conjunto.

implementación azul/verde

Estrategia de implementación en la que se crean dos entornos separados, pero idénticos. La versión actual de la aplicación se ejecuta en un entorno (azul) y la nueva versión de la aplicación se ejecuta en el otro entorno (verde). Esta estrategia lo ayuda a hacer reversiones rápidas con un impacto mínimo.

bot

Aplicación de software que ejecuta tareas automatizadas a través de Internet y simula la actividad o interacción humana. Algunos bots son útiles o beneficiosos, como los rastreadores web que indexan la información de Internet. Otros bots, conocidos como bots maliciosos, tienen como objetivo causar interrupciones o daños a personas u organizaciones.

botnet

Redes de bots infectadas por malware y que están bajo el control de una sola parte, conocida como pastor de bots u operador de bots. Las botnets son el mecanismo más conocido para escalar los bots y su impacto.

branch

Área contenida de un repositorio de código. La primera rama que se crea en un repositorio es la rama principal. Puede crear una rama nueva a partir de una rama existente y, a continuación, desarrollar características o corregir errores en la rama nueva. Una rama que se genera para crear una característica se denomina comúnmente rama de característica. Cuando la característica se encuentra lista para su lanzamiento, se vuelve a combinar la rama de característica con la rama principal. Para obtener más información, consulte Acerca de las sucursales (GitHub documentación).

acceso de emergencia

En circunstancias excepcionales y mediante un proceso aprobado, es una forma rápida de que un usuario pueda acceder a un Cuenta de AWS sitio al que normalmente no tiene permisos de acceso. Para más información, consulte el indicador Implement break-glass procedures en la guía de AWS Well-Architected.

estrategia de implementación sobre infraestructura existente

La infraestructura existente en su entorno. Al adoptar una estrategia de implementación sobre infraestructura existente para una arquitectura de sistemas, se diseña la arquitectura en función de las limitaciones de los sistemas y la infraestructura actuales. Si está ampliando la infraestructura existente, puede combinar las estrategias de implementación sobre infraestructuras existentes y de implementación desde cero.

caché de búfer

El área de memoria donde se almacenan los datos a los que se accede con más frecuencia.

capacidad empresarial

Lo que hace una empresa para generar valor (por ejemplo, ventas, servicio al cliente o marketing). Las arquitecturas de microservicios y las decisiones de desarrollo pueden estar impulsadas por las capacidades empresariales. Para obtener más información, consulte la sección Organizado en torno a las capacidades empresariales del documento técnico Ejecutar microservicios en contenedores en AWS.

planificación de la continuidad del negocio (BCP)

Plan que aborda el posible impacto de un evento disruptivo, como una migración a gran escala en las operaciones y permite a la empresa reanudar las operaciones rápidamente.

C

CAF

Consulte AWS Cloud Adoption Framework.

implementación canario

Lanzamiento lento e incremental de una versión para los usuarios finales. Cuando tenga mayor confianza en la nueva versión, la implementa y reemplaza la versión actual en su totalidad.

CCoE

Consulte Centro de excelencia en la nube.

CDC

Consulte captura de datos de cambios.

captura de datos de cambio (CDC)

Proceso de seguimiento de los cambios en un origen de datos, como una tabla de base de datos, y registro de los metadatos relacionados con el cambio. Puede utilizar los CDC para diversos fines, como auditar o replicar los cambios en un sistema de destino para mantener la sincronización.

ingeniería del caos

Introducción intencionada de fallos o eventos disruptivos para poner a prueba la resiliencia de un sistema. Puedes usar AWS Fault Injection Service (AWS FIS) para realizar experimentos que estresen tus AWS cargas de trabajo y evalúen su respuesta.

CI/CD

Consulte integración continua y entrega continua.

clasificación

Un proceso de categorización que permite generar predicciones. Los modelos de ML para problemas de clasificación predicen un valor discreto. Los valores discretos siempre son distintos entre sí. Por ejemplo, es posible que un modelo necesite evaluar si hay o no un automóvil en una imagen.

cifrado del cliente

Cifrado de datos localmente, antes de que el objetivo los Servicio de AWS reciba.

Centro de excelencia en la nube (CCoE)

Equipo multidisciplinario que impulsa los esfuerzos de adopción de la nube en toda la organización, incluido el desarrollo de las prácticas recomendadas en la nube, la movilización de recursos, el establecimiento de plazos de migración y la dirección de la organización durante las transformaciones a gran escala. Para obtener más información, consulte las publicaciones de CCo E en el blog de estrategia Nube de AWS empresarial.

computación en la nube

La tecnología en la nube que se utiliza normalmente para la administración de dispositivos de IoT y el almacenamiento de datos de forma remota. La computación en la nube suele estar relacionada con la tecnología de computación de periferia.

modelo operativo en la nube

En una organización de TI, el modelo operativo que se utiliza para crear, madurar y optimizar uno o más entornos de nube. Para obtener más información, consulte Creación de su modelo operativo de nube.

etapas de adopción de la nube

Las siguientes son las cuatro fases por las que suelen pasar las empresas cuando migran a la Nube de AWS:

- Proyecto: ejecución de algunos proyectos relacionados con la nube con fines de prueba de concepto y aprendizaje
- Fundamento: realizar inversiones fundamentales para escalar su adopción de la nube (p. ej., crear una landing zone, definir una CCo E, establecer un modelo de operaciones)

- Migración: migración de aplicaciones individuales
- Reinvención: optimización de productos y servicios e innovación en la nube

Stephen Orban definió estas etapas en la entrada del blog The Journey Toward Cloud-First & the Stages of Adoption en el blog Nube de AWS Enterprise Strategy. Para obtener información sobre su relación con la estrategia de AWS migración, consulte la guía de preparación para la migración.

CMDB

Consulte base de datos de administración de configuración.

repositorio de código

Una ubicación donde el código fuente y otros activos, como documentación, muestras y scripts, se almacenan y actualizan mediante procesos de control de versiones. Algunos repositorios en la nube comunes son GitHub o Bitbucket Cloud. Cada versión del código se denomina rama. En una estructura de microservicios, cada repositorio se encuentra dedicado a una única funcionalidad. Una sola canalización de CI/CD puede utilizar varios repositorios.

caché en frío

Una caché de búfer que está vacía no está bien poblada o contiene datos obsoletos o irrelevantes. Esto afecta al rendimiento, ya que la instancia de la base de datos debe leer desde la memoria principal o el disco, lo que es más lento que leer desde la memoria caché del búfer.

datos fríos

Datos a los que se accede con poca frecuencia y que suelen ser históricos. Al consultar este tipo de datos, normalmente se aceptan consultas lentas. Trasladar estos datos a niveles o clases de almacenamiento de menor rendimiento y menos costosos puede reducir los costos.

visión artificial (CV)

Campo de la IA que utiliza el machine learning para analizar y extraer información de formatos visuales, como imágenes y videos digitales. Por ejemplo, Amazon SageMaker AI proporciona algoritmos de procesamiento de imágenes para CV.

deriva de configuración

En el caso de una carga de trabajo, un cambio en la configuración con respecto al estado esperado. Podría provocar que la carga de trabajo deje de cumplir las normas y, por lo general, es gradual e involuntaria.

base de datos de administración de configuración (CMDB)

Repositorio que almacena y administra información sobre una base de datos y su entorno de TI, incluidos los componentes de hardware y software y sus configuraciones. Por lo general, los datos de una CMDB se utilizan en la etapa de detección y análisis de la cartera de productos durante la migración.

paquete de conformidad

Un conjunto de AWS Config reglas y medidas correctivas que puede reunir para personalizar sus controles de conformidad y seguridad. Puede implementar un paquete de conformidad como una entidad única en una región Cuenta de AWS y, o en una organización, mediante una plantilla YAML. Para obtener más información, consulta los paquetes de conformidad en la documentación. AWS Config

integración y entrega continuas (CI/CD)

El proceso de automatización de las etapas de origen, compilación, prueba, puesta en escena y producción del proceso de publicación del software. CI/CD se describe comúnmente como una canalización. CI/CD puede ayudarlo a automatizar los procesos, mejorar la productividad, mejorar la calidad del código y entregar más rápido. Para obtener más información, consulte Beneficios de la entrega continua. CD también puede significar implementación continua. Para obtener más información, consulte Entrega continua frente a implementación continua.

CV

Consulte visión artificial.

D

datos en reposo

Datos que están estacionarios en la red, como los datos que se encuentran almacenados.

clasificación de datos

Un proceso para identificar y clasificar los datos de su red en función de su importancia y sensibilidad. Es un componente fundamental de cualquier estrategia de administración de riesgos de ciberseguridad porque lo ayuda a determinar los controles de protección y retención adecuados para los datos. La clasificación de datos es un componente del pilar de seguridad del AWS Well-Architected Framework. Para obtener más información, consulte Clasificación de datos.

deriva de datos

Una variación significativa entre los datos de producción y los datos que se utilizaron para entrenar un modelo de machine learning, o un cambio significativo en los datos de entrada a lo largo del tiempo. La deriva de datos puede reducir la calidad, la precisión y la imparcialidad generales de las predicciones de los modelos de machine learning.

datos en tránsito

Datos que se mueven de forma activa por la red, por ejemplo, entre los recursos de la red.

malla de datos

Marco de arquitectura que proporciona una propiedad de datos distribuida y descentralizada con una administración y una gobernanza centralizadas.

minimización de datos

El principio de recopilar y procesar solo los datos estrictamente necesarios. Practicar la minimización de los datos Nube de AWS puede reducir los riesgos de privacidad, los costos y la huella de carbono de la analítica.

perímetro de datos

Un conjunto de barreras preventivas en su AWS entorno que ayudan a garantizar que solo las identidades confiables accedan a los recursos confiables desde las redes esperadas. Para obtener más información, consulte Crear un perímetro de datos sobre. AWS

preprocesamiento de datos

Transformar los datos sin procesar en un formato que su modelo de ML pueda analizar fácilmente. El preprocesamiento de datos puede implicar eliminar determinadas columnas o filas y corregir los valores faltantes, incoherentes o duplicados.

procedencia de los datos

El proceso de rastrear el origen y el historial de los datos a lo largo de su ciclo de vida, por ejemplo, la forma en que se generaron, transmitieron y almacenaron los datos.

titular de los datos

Persona cuyos datos se recopilan y procesan.

almacenamiento de datos

Sistema de administración de datos que respalda la inteligencia empresarial, como los análisis. Los almacenes de datos suelen contener grandes cantidades de datos históricos y, por lo general, se utilizan para las consultas y los análisis.

lenguaje de definición de datos (DDL)

Instrucciones o comandos para crear o modificar la estructura de tablas y objetos de una base de datos.

lenguaje de manipulación de datos (DML)

Instrucciones o comandos para modificar (insertar, actualizar y eliminar) la información de una base de datos.

DDL

Consulte lenguaje de definición de bases de datos.

conjunto profundo

Combinar varios modelos de aprendizaje profundo para la predicción. Puede utilizar conjuntos profundos para obtener una predicción más precisa o para estimar la incertidumbre de las predicciones.

aprendizaje profundo

Un subcampo del ML que utiliza múltiples capas de redes neuronales artificiales para identificar el mapeo entre los datos de entrada y las variables objetivo de interés.

defense-in-depth

Un enfoque de seguridad de la información en el que se distribuyen cuidadosamente una serie de mecanismos y controles de seguridad en una red informática para proteger la confidencialidad, la integridad y la disponibilidad de la red y de los datos que contiene. Al adoptar esta estrategia AWS, se añaden varios controles en diferentes capas de la AWS Organizations estructura para ayudar a proteger los recursos. Por ejemplo, un defense-in-depth enfoque podría combinar la autenticación multifactorial, la segmentación de la red y el cifrado.

administrador delegado

En AWS Organizations, un servicio compatible puede registrar una cuenta de AWS miembro para administrar las cuentas de la organización y gestionar los permisos de ese servicio. Esta

cuenta se denomina administrador delegado para ese servicio. Para obtener más información y una lista de servicios compatibles, consulte Servicios que funcionan con AWS Organizations en la documentación de AWS Organizations .

Implementación

El proceso de hacer que una aplicación, características nuevas o correcciones de código se encuentren disponibles en el entorno de destino. La implementación abarca implementar cambios en una base de código y, a continuación, crear y ejecutar esa base en los entornos de la aplicación.

entorno de desarrollo

Consulte entorno.

control de detección

Un control de seguridad que se ha diseñado para detectar, registrar y alertar después de que se produzca un evento. Estos controles son una segunda línea de defensa, ya que lo advierten sobre los eventos de seguridad que han eludido los controles preventivos establecidos. Para obtener más información, consulte Controles de detección en Implementación de controles de seguridad en AWS.

asignación de flujos de valor para el desarrollo (DVSM)

Proceso que se utiliza para identificar y priorizar las restricciones que afectan negativamente a la velocidad y la calidad en el ciclo de vida del desarrollo de software. DVSM amplía el proceso de asignación del flujo de valor diseñado originalmente para las prácticas de fabricación ajustada. Se centra en los pasos y los equipos necesarios para crear y transferir valor a través del proceso de desarrollo de software.

gemelo digital

Representación virtual de un sistema del mundo real, como un edificio, una fábrica, un equipo industrial o una línea de producción. Los gemelos digitales son compatibles con el mantenimiento predictivo, la supervisión remota y la optimización de la producción.

tabla de dimensiones

En un esquema en estrella, tabla más pequeña que contiene los atributos de datos sobre los datos cuantitativos en una tabla de hechos. Los atributos de la tabla de dimensiones suelen ser campos de texto o números discretos que se comportan como texto. Estos atributos se suelen utilizar para restringir consultas, filtrarlas y etiquetar los conjuntos de resultados.

desastre

Un evento que impide que una carga de trabajo o un sistema cumplan sus objetivos empresariales en su ubicación principal de implementación. Estos eventos pueden ser desastres naturales, fallos técnicos o el resultado de acciones humanas, como una configuración incorrecta involuntaria o un ataque de malware.

recuperación de desastres (DR)

Estrategia y proceso que utiliza para minimizar el tiempo de inactividad y la pérdida de datos a causa de un desastre. Para obtener más información, consulte Recuperación ante desastres de cargas de trabajo en AWS: Recovery in the Cloud in the AWS Well-Architected Framework.

DML

Consulte lenguaje de manipulación de bases de datos.

diseño basado en el dominio

Un enfoque para desarrollar un sistema de software complejo mediante la conexión de sus componentes a dominios en evolución, o a los objetivos empresariales principales, a los que sirve cada componente. Este concepto lo introdujo Eric Evans en su libro, Diseño impulsado por el dominio: abordando la complejidad en el corazón del software (Boston: Addison-Wesley Professional, 2003). Para obtener información sobre cómo utilizar el diseño basado en dominios con el patrón de higos estranguladores, consulte Modernización gradual de los servicios web antiguos de Microsoft ASP.NET (ASMX) mediante contenedores y Amazon API Gateway.

DR

Consulte recuperación ante desastres.

Detección de desviaciones

Seguimiento de las desviaciones con respecto a una configuración con línea de base. Por ejemplo, puedes usarlo AWS CloudFormation para detectar desviaciones en los recursos del sistema o puedes usarlo AWS Control Tower para detectar cambios en tu landing zone que puedan afectar al cumplimiento de los requisitos de gobierno.

DVSM

Consulte asignación de flujos de valor para el desarrollo.

E

EDA

Consulte análisis de datos de tipo exploratorio.

EDI

Consulte intercambio electrónico de datos.

computación en la periferia

La tecnología que aumenta la potencia de cálculo de los dispositivos inteligentes en la periferia de una red de IoT. En comparación con la computación en la nube, la computación de periferia puede reducir la latencia de la comunicación y mejorar el tiempo de respuesta.

intercambio electrónico de datos (EDI)

Intercambio automatizado de documentos comerciales entre organizaciones. Para más información, consulte ¿Qué es el intercambio electrónico de datos?

cifrado

Proceso de computación que transforma datos de texto plano, que son legibles por humanos, en texto cifrado.

clave de cifrado

Cadena criptográfica de bits aleatorios que se genera mediante un algoritmo de cifrado. Las claves pueden variar en longitud y cada una se ha diseñado para ser impredecible y única.

endianidad

El orden en el que se almacenan los bytes en la memoria del ordenador. Los sistemas big-endianos almacenan primero el byte más significativo. Los sistemas Little-Endian almacenan primero el byte menos significativo.

punto de conexión

Consulte punto de conexión de servicio.

servicio de punto de conexión

Servicio que puede alojar en una nube privada virtual (VPC) para compartir con otros usuarios. Puede crear un servicio de punto final AWS PrivateLink y conceder permisos a otros Cuentas de AWS o a responsables AWS Identity and Access Management (de IAM). Estas cuentas o

entidades principales pueden conectarse a su servicio de punto de conexión de forma privada mediante la creación de puntos de conexión de VPC de interfaz. Para obtener más información, consulte Creación de un servicio de punto de conexión en la documentación de Amazon Virtual Private Cloud (Amazon VPC).

planificación de recursos empresariales (ERP)

Sistema que automatiza y administra los procesos empresariales clave (como la contabilidad, MES y la administración de proyectos) de una empresa.

cifrado de sobre

El proceso de cifrar una clave de cifrado con otra clave de cifrado. Para obtener más información, consulte el cifrado de sobres en la documentación de AWS Key Management Service (AWS KMS).

entorno

Una instancia de una aplicación en ejecución. Los siguientes son los tipos de entornos más comunes en la computación en la nube:

- entorno de desarrollo: instancia de una aplicación en ejecución que solo se encuentra disponible para el equipo principal responsable del mantenimiento de la aplicación. Los entornos de desarrollo se utilizan para probar los cambios antes de promocionarlos a los entornos superiores. Este tipo de entorno a veces se denomina entorno de prueba.
- entornos inferiores: todos los entornos de desarrollo de una aplicación, como los que se utilizan para las compilaciones y pruebas iniciales.
- entorno de producción: instancia de una aplicación en ejecución a la que pueden acceder los usuarios finales. En un CI/CD proceso, el entorno de producción es el último entorno de implementación.
- entornos superiores: todos los entornos a los que pueden acceder usuarios que no sean del equipo de desarrollo principal. Esto puede incluir un entorno de producción, entornos de preproducción y entornos para las pruebas de aceptación por parte de los usuarios.

epopeya

En las metodologías ágiles, son categorías funcionales que ayudan a organizar y priorizar el trabajo. Las epopeyas brindan una descripción detallada de los requisitos y las tareas de implementación. Por ejemplo, las epopeyas AWS de seguridad de CAF incluyen la gestión de identidades y accesos, los controles de detección, la seguridad de la infraestructura, la protección de datos y la respuesta a incidentes. Para obtener más información sobre las epopeyas en la estrategia de migración de AWS , consulte la Guía de implementación del programa.

ERP

Consulte planificación de recursos empresariales.

análisis de datos de tipo exploratorio (EDA)

El proceso de analizar un conjunto de datos para comprender sus características principales. Se recopilan o agregan datos y, a continuación, se realizan las investigaciones iniciales para encontrar patrones, detectar anomalías y comprobar las suposiciones. El EDA se realiza mediante el cálculo de estadísticas resumidas y la creación de visualizaciones de datos.

F

tabla de hechos

Tabla central de un esquema en estrella. Almacena datos cuantitativos sobre operaciones empresariales. Por lo general, una tabla de hechos contiene dos tipos de columnas: las que contienen medidas y las que contienen una clave externa para una tabla de dimensiones.

Fail Fast

Filosofía que utiliza pruebas frecuentes e incrementales para reducir el ciclo de vida del desarrollo. Es una parte fundamental de los enfoques ágiles.

límite de aislamiento de errores

En el Nube de AWS, un límite, como una zona de disponibilidad Región de AWS, un plano de control o un plano de datos, que limita el efecto de una falla y ayuda a mejorar la resiliencia de las cargas de trabajo. Para más información, consulte AWS Fault Isolation Boundaries.

rama de característica

Consulte rama.

características

Los datos de entrada que se utilizan para hacer una predicción. Por ejemplo, en un contexto de fabricación, las características pueden ser imágenes que se capturan periódicamente desde la línea de fabricación.

importancia de las características

La importancia que tiene una característica para las predicciones de un modelo. Por lo general, esto se expresa como una puntuación numérica que se puede calcular mediante diversas

técnicas, como las explicaciones aditivas de Shapley (SHAP) y los gradientes integrados. Para obtener más información, consulte Interpretabilidad del modelo de aprendizaje automático con. AWS

transformación de funciones

Optimizar los datos para el proceso de ML, lo que incluye enriquecer los datos con fuentes adicionales, escalar los valores o extraer varios conjuntos de información de un solo campo de datos. Esto permite que el modelo de ML se beneficie de los datos. Por ejemplo, si divide la fecha del “27 de mayo de 2021 00:15:37” en “jueves”, “mayo”, “2021” y “15”, puede ayudar al algoritmo de aprendizaje a aprender patrones matizados asociados a los diferentes componentes de los datos.

peticiones con pocos pasos

Proporcionar a un LLM una pequeña cantidad de ejemplos que demuestren la tarea y el resultado deseado antes de pedirle que lleve a cabo una tarea similar. Esta técnica es una aplicación del aprendizaje contextual, mediante el que los modelos aprenden a partir de ejemplos (pasos) incrustados en las peticiones. La técnica de peticiones con pocos pasos puede ser eficaz para las tareas que requieren un formato, un razonamiento o un conocimiento del dominio específicos. Consulte también peticiones desde cero.

FGAC

Consulte control de acceso detallado.

control de acceso preciso (FGAC)

El uso de varias condiciones que tienen por objetivo permitir o denegar una solicitud de acceso.

migración relámpago

Método de migración de bases de datos que utiliza la replicación continua de datos mediante la captura de datos de cambio para migrar los datos en el menor tiempo posible, en lugar de utilizar un enfoque gradual. El objetivo es reducir al mínimo el tiempo de inactividad.

FM

Consulte modelo fundacional.

Modelo fundacional (FM)

Una gran red neuronal de aprendizaje profundo que se ha estado entrenando con conjuntos de datos masivos de datos generalizados y sin etiquetar. FMs son capaces de realizar una amplia variedad de tareas generales, como comprender el lenguaje, generar texto e imágenes

y conversar en lenguaje natural. Para más información, consulte ¿Qué son los modelos fundacionales?

G

IA generativa

Subconjunto de modelos de IA que se entrenaron con grandes cantidades de datos y que pueden utilizar una simple petición de texto para crear contenido y artefactos nuevos, como imágenes, videos, texto y audio. Para más información, consulte ¿Qué es la IA generativa?

bloqueo geográfico

Consulte restricciones geográficas.

restricciones geográficas (bloqueo geográfico)

En Amazon CloudFront, una opción para impedir que los usuarios de países específicos accedan a las distribuciones de contenido. Puede utilizar una lista de permitidos o bloqueados para especificar los países aprobados y prohibidos. Para obtener más información, consulta la sección Restringir la distribución geográfica del contenido en la CloudFront documentación.

Flujo de trabajo de Gitflow

Un enfoque en el que los entornos inferiores y superiores utilizan diferentes ramas en un repositorio de código fuente. El flujo de trabajo de Gitflow se considera heredado, mientras que el flujo de trabajo basado en enlaces troncales es el enfoque moderno preferido.

imagen dorada

Instantánea de un sistema o software que se usa como plantilla para implementar nuevas instancias de ese sistema o software. Por ejemplo, en la fabricación, una imagen dorada se puede utilizar para aprovisionar software en varios dispositivos y ayuda a mejorar la velocidad, la escalabilidad y la productividad de las operaciones de fabricación de dispositivos.

estrategia de implementación desde cero

La ausencia de infraestructura existente en un entorno nuevo. Al adoptar una estrategia de implementación desde cero para una arquitectura de sistemas, puede seleccionar todas las tecnologías nuevas sin que estas deban ser compatibles con una infraestructura existente, lo que también se conoce como implementación sobre infraestructura existente. Si está ampliando la infraestructura existente, puede combinar las estrategias de implementación sobre infraestructuras existentes y de implementación desde cero.

barrera de protección

Una regla de alto nivel que ayuda a regular los recursos, las políticas y el cumplimiento en todas las unidades organizativas (OUs). Las barreras de protección preventivas aplican políticas para garantizar la alineación con los estándares de conformidad. Se implementan mediante políticas de control de servicios y límites de permisos de IAM. Las barreras de protección de detección detectan las vulneraciones de las políticas y los problemas de conformidad, y generan alertas para su corrección. Se implementan mediante Amazon AWS Config AWS Security Hub CSPM GuardDuty AWS Trusted Advisor, Amazon Inspector y AWS Lambda cheques personalizados.

H

HA

Consulte alta disponibilidad.

migración heterogénea de bases de datos

Migración de la base de datos de origen a una base de datos de destino que utilice un motor de base de datos diferente (por ejemplo, de Oracle a Amazon Aurora). La migración heterogénea suele ser parte de un esfuerzo de rediseño de la arquitectura y convertir el esquema puede ser una tarea compleja. AWS ofrece AWS SCT, lo cual ayuda con las conversiones de esquemas.

alta disponibilidad (HA)

La capacidad de una carga de trabajo para funcionar de forma continua, sin intervención, en caso de desafíos o desastres. Los sistemas de alta disponibilidad están diseñados para realizar una conmutación por error automática, ofrecer un rendimiento de alta calidad de forma constante y gestionar diferentes cargas y fallos con un impacto mínimo en el rendimiento.

modernización histórica

Un enfoque utilizado para modernizar y actualizar los sistemas de tecnología operativa (TO) a fin de satisfacer mejor las necesidades de la industria manufacturera. Un histórico es un tipo de base de datos que se utiliza para recopilar y almacenar datos de diversas fuentes en una fábrica.

datos de reserva

Parte de los datos históricos etiquetados que se ocultan de un conjunto de datos que se utiliza para entrenar un modelo de machine learning. Puede utilizar los datos de reserva para evaluar el rendimiento del modelo mediante la comparación de las predicciones del modelo con los datos de reserva.

migración homogénea de bases de datos

Migración de la base de datos de origen a una base de datos de destino que comparte el mismo motor de base de datos (por ejemplo, Microsoft SQL Server a Amazon RDS para SQL Server). La migración homogénea suele formar parte de un esfuerzo para volver a alojar o redefinir la plataforma. Puede utilizar las utilidades de bases de datos nativas para migrar el esquema.

datos recientes

Datos a los que se accede con frecuencia, como datos en tiempo real o datos traslacionales recientes. Por lo general, estos datos requieren un nivel o una clase de almacenamiento de alto rendimiento para proporcionar respuestas rápidas a las consultas.

hotfix

Una solución urgente para un problema crítico en un entorno de producción. Debido a su urgencia, una revisión suele realizarse fuera del flujo de trabajo de DevOps publicación típico.

periodo de hiperatención

Periodo, inmediatamente después de la transición, durante el cual un equipo de migración administra y monitorea las aplicaciones migradas en la nube para solucionar cualquier problema. Por lo general, este periodo dura de 1 a 4 días. Al final del periodo de hiperatención, el equipo de migración suele transferir la responsabilidad de las aplicaciones al equipo de operaciones en la nube.

I

IaC

Consulte infraestructura como código.

políticas basadas en identidades

Política asociada a uno o más directores de IAM que define sus permisos en el entorno. Nube de AWS

aplicación inactiva

Aplicación que utiliza un promedio de CPU y memoria de entre 5 y 20 por ciento durante un periodo de 90 días. En un proyecto de migración, es habitual retirar estas aplicaciones o mantenerlas en las instalaciones.

IIoT

Consulte Internet de las cosas industrial.

infraestructura inmutable

Modelo que implementa una nueva infraestructura para las cargas de trabajo de producción en lugar de actualizar o modificar la infraestructura existente o aplicarle revisiones. Las infraestructuras inmutables son de manera intrínseca más coherentes, fiables y predecibles que las infraestructuras mutables. Para más información, consulte la práctica recomendada Implementación mediante una infraestructura inmutable en el Marco de AWS Well-Architected.

VPC entrante (de entrada)

En una arquitectura de AWS cuentas múltiples, una VPC que acepta, inspecciona y enruta las conexiones de red desde fuera de una aplicación. La arquitectura AWS de referencia de seguridad recomienda configurar la cuenta de red con entradas, salidas e inspección VPCs para proteger la interfaz bidireccional entre la aplicación y el resto de Internet.

migración gradual

Estrategia de transición en la que se migra la aplicación en partes pequeñas en lugar de realizar una transición única y completa. Por ejemplo, puede trasladar inicialmente solo unos pocos microservicios o usuarios al nuevo sistema. Tras comprobar que todo funciona correctamente, puede trasladar microservicios o usuarios adicionales de forma gradual hasta que pueda retirar su sistema heredado. Esta estrategia reduce los riesgos asociados a las grandes migraciones.

Industria 4.0

Término que introdujo Klaus Schwab en 2016 para referirse a la modernización de los procesos de fabricación mediante los avances en la conectividad, los datos en tiempo real, la automatización, el análisis, la IA y el ML.

infraestructura

Todos los recursos y activos que se encuentran en el entorno de una aplicación.

infraestructura como código (IaC)

Proceso de aprovisionamiento y administración de la infraestructura de una aplicación mediante un conjunto de archivos de configuración. La IaC se ha diseñado para ayudarlo a centralizar la administración de la infraestructura, estandarizar los recursos y escalar con rapidez a fin de que los entornos nuevos sean repetibles, fiables y consistentes.

Internet de las cosas industrial (T) IIo

El uso de sensores y dispositivos conectados a Internet en los sectores industriales, como el productivo, el eléctrico, el automotriz, el sanitario, el de las ciencias de la vida y el de la agricultura. Para obtener más información, consulte Creación de una estrategia de transformación digital de la Internet de las cosas (IIoT) industrial.

VPC de inspección

En una arquitectura de AWS cuentas múltiples, una VPC centralizada que gestiona las inspecciones del tráfico de red VPCs entre Internet y las redes locales (en una misma o Regiones de AWS diferente). La arquitectura AWS de referencia de seguridad recomienda configurar su cuenta de red con entrada, salida e inspección VPCs para proteger la interfaz bidireccional entre la aplicación e Internet en general.

Internet de las cosas (IoT)

Red de objetos físicos conectados con sensores o procesadores integrados que se comunican con otros dispositivos y sistemas a través de Internet o de una red de comunicación local. Para obtener más información, consulte ¿Qué es IoT?.

interpretabilidad

Característica de un modelo de machine learning que describe el grado en que un ser humano puede entender cómo las predicciones del modelo dependen de sus entradas. Para obtener más información, consulte Interpretabilidad del modelo de aprendizaje automático con. AWS

IoT

Consulte Internet de las cosas.

biblioteca de información de TI (ITIL)

Conjunto de prácticas recomendadas para ofrecer servicios de TI y alinearlos con los requisitos empresariales. La ITIL proporciona la base para la ITSM.

administración de servicios de TI (ITSM)

Actividades asociadas con el diseño, la implementación, la administración y el soporte de los servicios de TI para una organización. Para obtener información sobre la integración de las operaciones en la nube con las herramientas de ITSM, consulte la Guía de integración de operaciones.

ITIL

Consulte biblioteca de información de TI.

ITSM

Consulte administración de servicios de TI.

L

control de acceso basado en etiquetas (LBAC)

Una implementación del control de acceso obligatorio (MAC) en la que a los usuarios y a los propios datos se les asigna explícitamente un valor de etiqueta de seguridad. La intersección entre la etiqueta de seguridad del usuario y la etiqueta de seguridad de los datos determina qué filas y columnas puede ver el usuario.

zona de aterrizaje

Una landing zone es un AWS entorno multicuenta bien diseñado, escalable y seguro. Este es un punto de partida desde el cual las empresas pueden lanzar e implementar rápidamente cargas de trabajo y aplicaciones con confianza en su entorno de seguridad e infraestructura. Para obtener más información sobre las zonas de aterrizaje, consulte Configuración de un entorno de AWS seguro y escalable con varias cuentas.

modelo de lenguaje de gran tamaño (LLM)

Modelo de IA de aprendizaje profundo que se entrenó previamente con una gran cantidad de datos. Un LLM puede llevar a cabo varias tareas, como responder preguntas, resumir documentos, traducir textos a otros idiomas y completar oraciones. Para obtener más información, consulte Qué son. LLMs

migración grande

Migración de 300 servidores o más.

LBAC

Consulte control de acceso basado en etiquetas.

privilegio mínimo

La práctica recomendada de seguridad que consiste en conceder los permisos mínimos necesarios para realizar una tarea. Para obtener más información, consulte Aplicar permisos de privilegio mínimo en la documentación de IAM.

migrar mediante lift-and-shift

Consulte Las 7 R.

sistema little-endian

Un sistema que almacena primero el byte menos significativo. Consulte también endianidad.

LLM

Consulte modelo de lenguaje de gran tamaño.

entornos inferiores

Consulte entorno.

M

machine learning (ML)

Un tipo de inteligencia artificial que utiliza algoritmos y técnicas para el reconocimiento y el aprendizaje de patrones. El ML analiza y aprende de los datos registrados, como los datos del Internet de las cosas (IoT), para generar un modelo estadístico basado en patrones. Para más información, consulte Machine learning.

rama principal

Consulte rama.

malware

Software diseñado para comprometer la seguridad o la privacidad de la computadora. El malware podría interrumpir los sistemas informáticos, filtrar información confidencial u obtener acceso no autorizado. Algunos ejemplos de malware son los virus, los gusanos, el ransomware, los troyanos, el spyware y los registradores de pulsaciones de teclas.

Servicios administrados

Servicios de AWS para lo cual AWS opera la capa de infraestructura, el sistema operativo y las plataformas, y se accede a los puntos finales para almacenar y recuperar datos. Amazon Simple Storage Service (Amazon S3) y Amazon DynamoDB son ejemplos de servicios administrados. También se conocen como servicios abstractos.

sistema de ejecución de fabricación (MES)

Sistema de software para seguir, supervisar, documentar y controlar los procesos de producción que convierten las materias primas en productos acabados en la zona de producción.

MAP

Consulte Programa de aceleración de la migración.

mecanismo

Proceso completo mediante el que se crea una herramienta, se impulsa su adopción y, a continuación, se inspeccionan los resultados para hacer ajustes. Un mecanismo es un ciclo que se refuerza y mejora por sí mismo a medida que funciona. Para obtener más información, consulte Creación de mecanismos en el AWS Well-Architected Framework.

cuenta de miembro

Todas las Cuentas de AWS demás cuentas, excepto la de administración, que forman parte de una organización. AWS Organizations Una cuenta no puede pertenecer a más de una organización a la vez.

MES

Consulte sistema de ejecución de fabricación.

Message Queuing Telemetry Transport (MQTT)

Un protocolo de comunicación ligero machine-to-machine (M2M), basado en el patrón de publicación/suscripción, para dispositivos de IoT con recursos limitados.

microservicio

Un servicio pequeño e independiente que se comunica a través de una red bien definida APIs y que, por lo general, es propiedad de equipos pequeños e independientes. Por ejemplo, un sistema de seguros puede incluir microservicios que se adapten a las capacidades empresariales, como las de ventas o marketing, o a subdominios, como las de compras, reclamaciones o análisis. Los beneficios de los microservicios incluyen la agilidad, la escalabilidad flexible, la facilidad de implementación, el código reutilizable y la resiliencia. Para obtener más información, consulte Integrar microservicios mediante AWS servicios sin servidor.

arquitectura de microservicios

Un enfoque para crear una aplicación con componentes independientes que ejecutan cada proceso de la aplicación como un microservicio. Estos microservicios se comunican a través de una interfaz bien definida mediante un uso ligero. APIs Cada microservicio de esta arquitectura se puede actualizar, implementar y escalar para satisfacer la demanda de funciones específicas de una aplicación. Para obtener más información, consulte Implementación de microservicios en. AWS

Programa de aceleración de la migración (MAP)

Un AWS programa que proporciona soporte de consultoría, formación y servicios para ayudar a las organizaciones a crear una base operativa sólida para migrar a la nube y para ayudar a compensar el costo inicial de las migraciones. El MAP incluye una metodología de migración para ejecutar las migraciones antiguas de forma metódica y un conjunto de herramientas para automatizar y acelerar los escenarios de migración más comunes.

migración a escala

Proceso de transferencia de la mayoría de la cartera de aplicaciones a la nube en oleadas, con más aplicaciones desplazadas a un ritmo más rápido en cada oleada. En esta fase, se utilizan las prácticas recomendadas y las lecciones aprendidas en las fases anteriores para implementar una fábrica de migración de equipos, herramientas y procesos con el fin de agilizar la migración de las cargas de trabajo mediante la automatización y la entrega ágil. Esta es la tercera fase de la estrategia de migración de AWS.

fábrica de migración

Equipos multifuncionales que agilizan la migración de las cargas de trabajo mediante enfoques automatizados y ágiles. Los equipos de las fábricas de migración suelen incluir a analistas y propietarios de operaciones, empresas, ingenieros de migración, desarrolladores y DevOps profesionales que trabajan a pasos agigantados. Entre el 20 y el 50 por ciento de la cartera de aplicaciones empresariales se compone de patrones repetidos que pueden optimizarse mediante un enfoque de fábrica. Para obtener más información, consulte la discusión sobre las fábricas de migración y la Guía de fábricas de migración a la nube en este contenido.

metadatos de migración

Información sobre la aplicación y el servidor que se necesita para completar la migración. Cada patrón de migración requiere un conjunto diferente de metadatos de migración. Algunos ejemplos de metadatos de migración son la subred de destino, el grupo de seguridad y AWS la cuenta.

patrón de migración

Tarea de migración repetible que detalla la estrategia de migración, el destino de la migración y la aplicación o el servicio de migración utilizados. Ejemplo: rehospede la migración a Amazon EC2 AWS con Application Migration Service.

Migration Portfolio Assessment (MPA)

Herramienta en línea que proporciona información a fin de validar los argumentos comerciales necesarios para migrar a la Nube de AWS. La MPA ofrece una evaluación detallada de la cartera

(adecuación del tamaño de los servidores, precios, comparaciones del costo total de propiedad, análisis de los costos de migración), así como una planificación de la migración (análisis y recopilación de datos de aplicaciones, agrupación de aplicaciones, priorización de la migración y planificación de oleadas). La herramienta MPA (requiere iniciar sesión) está disponible de forma gratuita para todos los AWS consultores y consultores de los socios de APN.

Evaluación de la preparación para la migración (MRA)

Proceso que consiste en obtener información sobre el estado de preparación de una organización para la nube, identificar sus puntos fuertes y débiles y elaborar un plan de acción para cerrar las brechas identificadas mediante el AWS CAF. Para obtener más información, consulte la Guía de preparación para la migración. La MRA es la primera fase de la estrategia de migración de AWS.

estrategia de migración

Enfoque utilizado para migrar una carga de trabajo a la Nube de AWS. Para más información, consulte la entrada Las 7 R de este glosario y también Mobilize your organization to accelerate large-scale migrations.

ML

Consulte machine learning.

modernización

Transformar una aplicación obsoleta (antigua o monolítica) y su infraestructura en un sistema ágil, elástico y de alta disponibilidad en la nube para reducir los gastos, aumentar la eficiencia y aprovechar las innovaciones. Para más información, consulte Strategy for modernizing applications in the Nube de AWS.

evaluación de la preparación para la modernización

Evaluación que ayuda a determinar la preparación para la modernización de las aplicaciones de una organización; identifica los beneficios, los riesgos y las dependencias; y determina qué tan bien la organización puede soportar el estado futuro de esas aplicaciones. El resultado de la evaluación es un esquema de la arquitectura objetivo, una hoja de ruta que detalla las fases de desarrollo y los hitos del proceso de modernización y un plan de acción para abordar las brechas identificadas. Para más información, consulte Evaluating modernization readiness for applications in the Nube de AWS.

aplicaciones monolíticas (monolitos)

Aplicaciones que se ejecutan como un único servicio con procesos estrechamente acoplados. Las aplicaciones monolíticas presentan varios inconvenientes. Si una característica de la

aplicación experimenta un aumento en la demanda, se debe escalar toda la arquitectura. Agregar o mejorar las características de una aplicación monolítica también se vuelve más complejo a medida que crece la base de código. Para solucionar problemas con la aplicación, puede utilizar una arquitectura de microservicios. Para obtener más información, consulte Descomposición de monolitos en microservicios.

MPA

Consulte Migration Portfolio Assessment.

MQTT

Consulte Message Queuing Telemetry Transport.

clasificación multiclase

Un proceso que ayuda a generar predicciones para varias clases (predice uno de más de dos resultados). Por ejemplo, un modelo de ML podría preguntar "¿Este producto es un libro, un automóvil o un teléfono?" o "¿Qué categoría de productos es más interesante para este cliente?".

infraestructura mutable

Modelo que actualiza y modifica la infraestructura actual para las cargas de trabajo de producción. Para mejorar la coherencia, la fiabilidad y la previsibilidad, el AWS Well-Architected Framework recomienda el uso de una infraestructura inmutable como práctica recomendada.

O

OAC

Consulte control de acceso de origen.

OAI

Consulte identidad de acceso de origen.

OCM

Consulte administración del cambio organizacional.

migración fuera de línea

Método de migración en el que la carga de trabajo de origen se elimina durante el proceso de migración. Este método implica un tiempo de inactividad prolongado y, por lo general, se utiliza para cargas de trabajo pequeñas y no críticas.

OI

Consulte integración de operaciones.

OLA

Consulte acuerdo de nivel operativo.

migración en línea

Método de migración en el que la carga de trabajo de origen se copia al sistema de destino sin que se desconecte. Las aplicaciones que están conectadas a la carga de trabajo pueden seguir funcionando durante la migración. Este método implica un tiempo de inactividad nulo o mínimo y, por lo general, se utiliza para cargas de trabajo de producción críticas.

OPC-UA

Consulte Open Process Communications: arquitectura unificada.

Open Process Communications: arquitectura unificada (OPC-UA)

Un protocolo de machine-to-machine comunicación (M2M) para la automatización industrial. OPC-UA establece un estándar de interoperabilidad con esquemas de autenticación, autorización y cifrado de datos.

acuerdo de nivel operativo (OLA)

Acuerdo que aclara lo que los grupos de TI operativos se comprometen a ofrecerse entre sí, para respaldar un acuerdo de nivel de servicio (SLA).

revisión de la preparación operativa (ORR)

Lista de comprobación de preguntas y prácticas recomendadas asociadas que son útiles para comprender, evaluar, prevenir o reducir el alcance de los incidentes y posibles errores. Para más información, consulte Operational Readiness Reviews (ORR) en el Marco de AWS Well-Architected.

tecnología operativa (TO)

Sistemas de hardware y software que funcionan con el entorno físico para controlar las operaciones, los equipos y la infraestructura industriales. En el sector de la fabricación, la integración de los sistemas de TO y tecnología de la información (TI) es un enfoque clave para las transformaciones de la industria 4.0.

integración de operaciones (OI)

Proceso de modernización de las operaciones en la nube, que implica la planificación de la preparación, la automatización y la integración. Para obtener más información, consulte la Guía de integración de las operaciones.

registro de seguimiento organizativo

Un registro creado por y AWS CloudTrail que registra todos los eventos para todos los miembros Cuentas de AWS de una organización. AWS Organizations Este registro de seguimiento se crea en cada Cuenta de AWS que forma parte de la organización y realiza un seguimiento de la actividad en cada cuenta. Para obtener más información, consulte Crear un registro para una organización en la CloudTrail documentación.

administración del cambio organizacional (OCM)

Marco para administrar las transformaciones empresariales importantes y disruptivas desde la perspectiva de las personas, la cultura y el liderazgo. La OCM ayuda a las empresas a prepararse para nuevos sistemas y estrategias y a realizar la transición a ellos, al acelerar la adopción de cambios, abordar los problemas de transición e impulsar cambios culturales y organizacionales. En la estrategia de AWS migración, este marco se denomina aceleración de personal, debido a la velocidad de cambio que requieren los proyectos de adopción de la nube. Para obtener más información, consulte la Guía de OCM.

control de acceso de origen (OAC)

En CloudFront, una opción mejorada para restringir el acceso y proteger el contenido del Amazon Simple Storage Service (Amazon S3). El OAC admite todos los buckets de S3 Regiones de AWS, el cifrado del lado del servidor AWS KMS (SSE-KMS) y las solicitudes dinámicas `PUT` y `DELETE` dirigidas al bucket de S3.

identidad de acceso de origen (OAI)

En CloudFront, una opción para restringir el acceso y proteger el contenido de Amazon S3. Cuando utiliza OAI, CloudFront crea un principal con el que Amazon S3 puede autenticarse. Los directores autenticados solo pueden acceder al contenido de un bucket de S3 a través de una distribución específica. CloudFront Consulte también el OAC, que proporciona un control de acceso más detallado y mejorado.

ORR

Consulte revisión de la preparación operativa.

OT

Consulte tecnología operativa.

VPC saliente (de salida)

En una arquitectura de AWS cuentas múltiples, una VPC que gestiona las conexiones de red que se inician desde una aplicación. La arquitectura AWS de referencia de seguridad recomienda configurar la cuenta de red con entradas, salidas e inspección VPCs para proteger la interfaz bidireccional entre la aplicación e Internet en general.

P

límite de permisos

Una política de administración de IAM que se adjunta a las entidades principales de IAM para establecer los permisos máximos que puede tener el usuario o el rol. Para obtener más información, consulte Límites de permisos en la documentación de IAM.

información de identificación personal (PII)

Información que, vista directamente o combinada con otros datos relacionados, puede utilizarse para deducir de manera razonable la identidad de una persona. Algunos ejemplos de información de identificación personal son los nombres, las direcciones y la información de contacto.

PII

Consulte información de identificación personal.

manual de estrategias

Conjunto de pasos predefinidos que capturan el trabajo asociado a las migraciones, como la entrega de las funciones de operaciones principales en la nube. Un manual puede adoptar la forma de scripts, manuales de procedimientos automatizados o resúmenes de los procesos o pasos necesarios para operar un entorno modernizado.

PLC

Consulte controlador lógico programable.

PLM

Consulte administración del ciclo de vida del producto.

policy

Objeto que puede definir permisos (consulte política basada en identidad), especificar las condiciones de acceso (consulte política basada en recursos) o definir los permisos máximos para todas las cuentas de una organización de AWS Organizations (consulte política de control de servicio).

persistencia políglota

Elegir de forma independiente la tecnología de almacenamiento de datos de un microservicio en función de los patrones de acceso a los datos y otros requisitos. Si sus microservicios tienen la misma tecnología de almacenamiento de datos, pueden enfrentarse a desafíos de implementación o experimentar un rendimiento deficiente. Los microservicios se implementan más fácilmente y logran un mejor rendimiento y escalabilidad si utilizan el almacén de datos que mejor se adapte a sus necesidades.

evaluación de cartera

Proceso de detección, análisis y priorización de la cartera de aplicaciones para planificar la migración. Para obtener más información, consulte la Evaluación de la preparación para la migración.

predicate

Condición de consulta que devuelve `true` o `false`. En general, se encuentra en una cláusula `WHERE`.

inserción de predicados

Técnica de optimización de consultas en bases de datos que filtra los datos de la consulta antes de transferirlos. Esta técnica reduce la cantidad de datos de la base de datos relacional que se tienen que recuperar y procesar. Además, mejora el rendimiento de las consultas.

control preventivo

Un control de seguridad diseñado para evitar que ocurra un evento. Estos controles son la primera línea de defensa para evitar el acceso no autorizado o los cambios no deseados en la red. Para obtener más información, consulte Controles preventivos en Implementación de controles de seguridad en AWS.

entidad principal

Una entidad AWS que puede realizar acciones y acceder a los recursos. Esta entidad suele ser un usuario raíz para un Cuenta de AWS rol de IAM o un usuario. Para obtener más información, consulte Entidad principal en Términos y conceptos de roles en la documentación de IAM.

Privacidad desde el diseño

Enfoque de ingeniería de sistemas que tiene en cuenta la privacidad durante todo el proceso de desarrollo.

zonas alojadas privadas

Un contenedor que contiene información sobre cómo desea que Amazon Route 53 responda a las consultas de DNS de un dominio y sus subdominios dentro de uno o más VPCs. Para obtener más información, consulte Uso de zonas alojadas privadas en la documentación de Route 53.

control proactivo

Control de seguridad que se diseñó para evitar la implementación de recursos que no cumplan con la normativa. Estos controles analizan los recursos antes de aprovisionarlos. Si el recurso no cumple con los requisitos del control, no se aprovisiona. Para obtener más información, consulte la guía de referencia de controles en la AWS Control Tower documentación y consulte Controles proactivos en la sección Implementación de controles de seguridad en AWS.

administración del ciclo de vida del producto (PLM)

Administración de los datos y los procesos de un producto a lo largo de todo su ciclo de vida, desde el diseño, el desarrollo y el lanzamiento, pasando por el crecimiento y la madurez, hasta la reducción de su uso y su retirada.

entorno de producción

Consulte entorno.

controlador lógico programable (PLC)

En el sector de la fabricación, computadora adaptable y altamente fiable que supervisa las máquinas y automatiza los procesos de fabricación.

encadenamiento de peticiones

Uso de la salida de una petición de LLM como entrada para la siguiente petición a fin de generar mejores respuestas. Esta técnica se utiliza para dividir una tarea compleja en tareas secundarias o para refinar o ampliar de forma iterativa una respuesta preliminar. Ayuda a mejorar la precisión y la relevancia de las respuestas de un modelo y permite obtener resultados más detallados y personalizados.

seudonimización

El proceso de reemplazar los identificadores personales de un conjunto de datos por valores de marcadores de posición. La seudonimización puede ayudar a proteger la privacidad personal. Los datos seudonimizados siguen considerándose datos personales.

publish/subscribe (pub/sub)

Patrón que permite establecer comunicaciones asíncronas entre microservicios para mejorar la escalabilidad y la capacidad de respuesta. Por ejemplo, en un MES basado en microservicios, un microservicio puede publicar mensajes de eventos en un canal al que se pueden suscribir otros microservicios. El sistema puede agregar nuevos microservicios sin cambiar el servicio de publicación.

Q

plan de consulta

Serie de pasos, como instrucciones, que se utilizan para acceder a los datos de un sistema de base de datos relacional SQL.

regresión del plan de consulta

El optimizador de servicios de la base de datos elige un plan menos óptimo que antes de un cambio determinado en el entorno de la base de datos. Los cambios en estadísticas, restricciones, configuración del entorno, enlaces de parámetros de consultas y actualizaciones del motor de base de datos PostgreSQL pueden provocar una regresión del plan.

R

Matriz RACI

Consulte responsable, fiable, consultada e informada (RACI).

RAG

Consulte generación aumentada por recuperación.

ransomware

Software malicioso que se ha diseñado para bloquear el acceso a un sistema informático o a los datos hasta que se efectúe un pago.

Matriz RASCI

Consulte responsable, fiable, consultada e informada (RACI).

RCAC

Consulte control de acceso por filas y columnas.

réplica de lectura

Una copia de una base de datos que se utiliza con fines de solo lectura. Puede enrutar las consultas a la réplica de lectura para reducir la carga en la base de datos principal.

rediseñar

Consulte Las 7 R.

objetivo de punto de recuperación (RPO)

La cantidad de tiempo máximo aceptable desde el último punto de recuperación de datos. Esto determina qué se considera una pérdida de datos aceptable entre el último punto de recuperación y la interrupción del servicio.

objetivo de tiempo de recuperación (RTO)

La demora máxima aceptable entre la interrupción del servicio y el restablecimiento del servicio.

refactorizar

Consulte Las 7 R.

Region

Conjunto de AWS recursos en un área geográfica. Cada uno Región de AWS está aislado e independiente de los demás para proporcionar tolerancia a las fallas, estabilidad y resiliencia. Para más información, consulte Specify which Regiones de AWS your account can use.

regresión

Una técnica de ML que predice un valor numérico. Por ejemplo, para resolver el problema de "¿A qué precio se venderá esta casa?", un modelo de ML podría utilizar un modelo de regresión lineal para predecir el precio de venta de una vivienda en función de datos conocidos sobre ella (por ejemplo, los metros cuadrados).

volver a alojar

Consulte Las 7 R.

versión

En un proceso de implementación, el acto de promover cambios en un entorno de producción.

reubicar

Consulte Las 7 R.

redefinir la plataforma

Consulte Las 7 R.

recomprar

Consulte Las 7 R.

resiliencia

Capacidad de una aplicación para resistir interrupciones o recuperarse de ellas. Al planificar la resiliencia en la Nube de AWS, la alta disponibilidad y la recuperación ante desastres son consideraciones comunes. Para más información, consulte Resiliencia en la Nube de AWS.

política basada en recursos

Una política asociada a un recurso, como un bucket de Amazon S3, un punto de conexión o una clave de cifrado. Este tipo de política especifica a qué entidades principales se les permite el acceso, las acciones compatibles y cualquier otra condición que deba cumplirse.

matriz responsable, confiable, consultada e informada (RACI)

Una matriz que define las funciones y responsabilidades de todas las partes involucradas en las actividades de migración y las operaciones de la nube. El nombre de la matriz se deriva de los tipos de responsabilidad definidos en la matriz: responsable (R), contable (A), consultado (C) e informado (I). El tipo de soporte (S) es opcional. Si incluye el soporte, la matriz se denomina matriz RASCI y, si la excluye, se denomina matriz RACI.

control receptivo

Un control de seguridad que se ha diseñado para corregir los eventos adversos o las desviaciones con respecto a su base de seguridad. Para obtener más información, consulte Controles receptivos en Implementación de controles de seguridad en AWS.

retain

Consulte Las 7 R.

retirar

Consulte Las 7 R.

Generación aumentada de recuperación (RAG)

Tecnología de IA generativa mediante la que un LLM hace referencia a un origen de datos autorizado que se encuentra fuera de sus orígenes de datos de entrenamiento antes de generar una respuesta. Por ejemplo, un modelo de RAG podría hacer una búsqueda semántica en la base de conocimientos o en los datos personalizados de una organización. Para más información, consulte ¿Qué es RAG (generación aumentada por recuperación)?

rotación

Proceso mediante el que periódicamente se actualiza un secreto para que resulte más difícil que un atacante pueda acceder a las credenciales.

control de acceso por filas y columnas (RCAC)

El uso de expresiones SQL básicas y flexibles que tienen reglas de acceso definidas. El RCAC consta de permisos de fila y máscaras de columnas.

RPO

Consulte objetivo de punto de recuperación.

RTO

Consulte objetivo de tiempo de recuperación.

manual de procedimientos

Conjunto de procedimientos manuales o automatizados necesarios para realizar una tarea específica. Por lo general, se diseñan para agilizar las operaciones o los procedimientos repetitivos con altas tasas de error.

S

SAML 2.0

Un estándar abierto que utilizan muchos proveedores de identidad (IdPs). Esta función permite el inicio de sesión único (SSO) federado, de modo que los usuarios pueden iniciar sesión Consola de administración de AWS o llamar a las operaciones de la AWS API sin tener que crear un

usuario en IAM para todos los miembros de la organización. Para obtener más información sobre la federación basada en SAML 2.0, consulte Acerca de la federación basada en SAML 2.0 en la documentación de IAM.

SCADA

Consulte control de supervisión y adquisición de datos.

SCP

Consulte política de control de servicio.

secreta

En AWS Secrets Manager, información confidencial o restringida, como una contraseña o credenciales de usuario, que se almacena de forma cifrada. Se compone del valor del secreto y de sus metadatos. El valor del secreto puede ser binario, una sola cadena o varias cadenas. Para más información, consulte What's in a Secrets Manager secret? en la documentación de Secrets Manager.

seguridad desde el diseño

Enfoque de ingeniería de sistemas que tiene en cuenta la seguridad durante todo el proceso de desarrollo.

control de seguridad

Barrera de protección técnica o administrativa que impide, detecta o reduce la capacidad de un agente de amenazas para aprovechar una vulnerabilidad de seguridad. Existen cuatro tipos de controles de seguridad principales: preventivos, de detección, de respuesta y proactivos.

refuerzo de la seguridad

Proceso de reducir la superficie expuesta a ataques para hacerla más resistente a los ataques. Esto puede incluir acciones, como la eliminación de los recursos que ya no se necesitan, la implementación de prácticas recomendadas de seguridad consistente en conceder privilegios mínimos o la desactivación de características innecesarias en los archivos de configuración.

sistema de información sobre seguridad y administración de eventos (SIEM)

Herramientas y servicios que combinan sistemas de administración de información sobre seguridad (SIM) y de administración de eventos de seguridad (SEM). Un sistema de SIEM recopila, monitorea y analiza los datos de servidores, redes, dispositivos y otras fuentes para detectar amenazas y brechas de seguridad y generar alertas.

automatización de la respuesta de seguridad

Acción predefinida y programada que está diseñada para responder automáticamente a un evento de seguridad o corregirlo. Estas automatizaciones sirven como controles de seguridad preventivos o adaptables que le ayudan a implementar las mejores prácticas AWS de seguridad. La modificación de un grupo de seguridad de VPC, la aplicación de revisiones a una instancia de Amazon EC2 o la rotación de credenciales son algunos ejemplos de acciones de respuesta automatizadas.

cifrado del servidor

Cifrado de los datos en su destino, por parte de Servicio de AWS quien los recibe.

política de control de servicio (SCP)

Política que proporciona un control centralizado de los permisos de todas las cuentas de una organización en AWS Organizations. SCPs defina barreras o establezca límites a las acciones que un administrador puede delegar en usuarios o roles. Puede utilizarlas SCPs como listas de permitidos o rechazados para especificar qué servicios o acciones están permitidos o prohibidos. Para obtener más información, consulte las políticas de control de servicios en la AWS Organizations documentación.

punto de enlace de servicio

La URL del punto de entrada de un Servicio de AWS. Para conectarse mediante programación a un servicio de destino, puede utilizar un punto de conexión. Para obtener más información, consulte Puntos de conexión de Servicio de AWS en Referencia general de AWS.

acuerdo de nivel de servicio (SLA)

Acuerdo que aclara lo que un equipo de TI se compromete a ofrecer a los clientes, como el tiempo de actividad y el rendimiento del servicio.

indicador de nivel de servicio (SLI)

Medición de un aspecto del rendimiento de un servicio, como la tasa de errores, la disponibilidad o el rendimiento.

objetivo de nivel de servicio (SLO)

Métrica objetivo que representa el estado de un servicio medido mediante un indicador de nivel de servicio.

modelo de responsabilidad compartida

Un modelo que describe la responsabilidad con AWS la que compartes la seguridad y el cumplimiento de la nube. AWS es responsable de la seguridad de la nube, mientras que usted es responsable de la seguridad en la nube. Para obtener más información, consulte el Modelo de responsabilidad compartida.

SIEM

Consulte sistema de administración de eventos e información de seguridad.

único punto de error (SPOF)

Error en un único componente crítico de una aplicación que puede interrumpir el sistema.

SLA

Consulte acuerdo de nivel de servicio.

SLI

Consulte indicador de nivel de servicio.

SLO

Consulte objetivo de nivel de servicio.

split-and-seed modelo

Un patrón para escalar y acelerar los proyectos de modernización. A medida que se definen las nuevas funciones y los lanzamientos de los productos, el equipo principal se divide para crear nuevos equipos de productos. Esto ayuda a ampliar las capacidades y los servicios de su organización, mejora la productividad de los desarrolladores y apoya la innovación rápida. Para más información, consulte Phased approach to modernizing applications in the Nube de AWS.

SPOF

Consulte único punto de error.

esquema en estrella

Estructura organizativa de una base de datos que utiliza una tabla de hechos de gran tamaño para almacenar datos transaccionales o medidos y una o varias tablas dimensionales más pequeñas para almacenar los atributos de los datos. Esta estructura está diseñada para utilizarse en un almacén de datos o con fines de inteligencia empresarial.

patrón de higo estrangulador

Un enfoque para modernizar los sistemas monolíticos mediante la reescritura y el reemplazo gradual de las funciones del sistema hasta que se pueda desmantelar el sistema heredado. Este patrón utiliza la analogía de una higuera que crece hasta convertirse en un árbol estable y, finalmente, se apodera y reemplaza a su host. El patrón fue presentado por Martin Fowler como una forma de gestionar el riesgo al reescribir sistemas monolíticos. Para ver un ejemplo con la aplicación de este patrón, consulte Modernización gradual de los servicios web antiguos de Microsoft ASP.NET (ASMX) mediante contenedores y Amazon API Gateway.

subred

Un intervalo de direcciones IP en la VPC. Una subred debe residir en una sola zona de disponibilidad.

control de supervisión y adquisición de datos (SCADA)

En el sector de la fabricación, sistema que utiliza hardware y software para supervisar los activos físicos y las operaciones de producción.

cifrado simétrico

Un algoritmo de cifrado que utiliza la misma clave para cifrar y descifrar los datos.

pruebas sintéticas

Prueba de un sistema de manera que simule las interacciones de los usuarios para detectar posibles problemas o supervisar el rendimiento. Puede usar Amazon CloudWatch Synthetics para crear estas pruebas.

petición del sistema

Técnica para proporcionar contexto, instrucciones o pautas a un LLM para dirigir su comportamiento. Las peticiones del sistema ayudan a establecer el contexto y las reglas para las interacciones con los usuarios.

T

etiquetas

Pares clave-valor que actúan como metadatos para organizar los recursos. AWS Las etiquetas pueden ayudar a administrar, identificar, organizar, buscar y filtrar recursos de . Para obtener más información, consulte Etiquetado de los recursos de AWS.

variable de destino

El valor que intenta predecir en el ML supervisado. Esto también se conoce como variable de resultado. Por ejemplo, en un entorno de fabricación, la variable objetivo podría ser un defecto del producto.

lista de tareas

Herramienta que se utiliza para hacer un seguimiento del progreso mediante un manual de procedimientos. La lista de tareas contiene una descripción general del manual de procedimientos y una lista de las tareas generales que deben completarse. Para cada tarea general, se incluye la cantidad estimada de tiempo necesario, el propietario y el progreso.

entorno de prueba

Consulte entorno.

entrenamiento

Proporcionar datos de los que pueda aprender su modelo de ML. Los datos de entrenamiento deben contener la respuesta correcta. El algoritmo de aprendizaje encuentra patrones en los datos de entrenamiento que asignan los atributos de los datos de entrada al destino (la respuesta que desea predecir). Genera un modelo de ML que captura estos patrones. Luego, el modelo de ML se puede utilizar para obtener predicciones sobre datos nuevos para los que no se conoce el destino.

puerta de enlace de tránsito

Un centro de tránsito de red que puede usar para interconectar sus redes con VPCs las locales. Para obtener más información, consulte Qué es una pasarela de tránsito en la AWS Transit Gateway documentación.

flujo de trabajo basado en enlaces troncales

Un enfoque en el que los desarrolladores crean y prueban características de forma local en una rama de característica y, a continuación, combinan esos cambios en la rama principal. Luego, la rama principal se adapta a los entornos de desarrollo, preproducción y producción, de forma secuencial.

acceso de confianza

Otorgar permisos a un servicio que especifique para realizar tareas en su organización AWS Organizations y en sus cuentas en su nombre. El servicio de confianza crea un rol vinculado al servicio en cada cuenta, cuando ese rol es necesario, para realizar las tareas de administración

por usted. Para obtener más información, consulte AWS Organizations Utilización con otros AWS servicios en la AWS Organizations documentación.

ajuste

Cambiar aspectos de su proceso de formación a fin de mejorar la precisión del modelo de ML. Por ejemplo, puede entrenar el modelo de ML al generar un conjunto de etiquetas, incorporar etiquetas y, luego, repetir estos pasos varias veces con diferentes ajustes para optimizar el modelo.

equipo de dos pizzas

Un DevOps equipo pequeño al que puedes alimentar con dos pizzas. Un equipo formado por dos integrantes garantiza la mejor oportunidad posible de colaboración en el desarrollo de software.

U

incertidumbre

Un concepto que hace referencia a información imprecisa, incompleta o desconocida que puede socavar la fiabilidad de los modelos predictivos de ML. Hay dos tipos de incertidumbre: la incertidumbre epistémica se debe a datos limitados e incompletos, mientras que la incertidumbre aleatoria se debe al ruido y la aleatoriedad inherentes a los datos. Para más información, consulte la guía Cuantificación de la incertidumbre en los sistemas de aprendizaje profundo.

tareas indiferenciadas

También conocido como tareas arduas, es el trabajo que es necesario para crear y operar una aplicación, pero que no proporciona un valor directo al usuario final ni proporciona una ventaja competitiva. Algunos ejemplos de tareas indiferenciadas son la adquisición, el mantenimiento y la planificación de la capacidad.

entornos superiores

Consulte entorno.

V

succión

Una operación de mantenimiento de bases de datos que implica limpiar después de las actualizaciones incrementales para recuperar espacio de almacenamiento y mejorar el rendimiento.

control de versión

Procesos y herramientas que realizan un seguimiento de los cambios, como los cambios en el código fuente de un repositorio.

Emparejamiento de VPC

Una conexión entre dos VPCs que le permite enrutar el tráfico mediante direcciones IP privadas. Para obtener más información, consulte ¿Qué es una interconexión de VPC? en la documentación de Amazon VPC.

vulnerabilidad

Defecto de software o hardware que pone en peligro la seguridad del sistema.

W

caché caliente

Un búfer caché que contiene datos actuales y relevantes a los que se accede con frecuencia. La instancia de base de datos puede leer desde la caché del búfer, lo que es más rápido que leer desde la memoria principal o el disco.

datos templados

Datos a los que el acceso es infrecuente. Al consultar este tipo de datos, normalmente se aceptan consultas moderadamente lentas.

función de ventana

Función SQL que hace un cálculo en un grupo de filas que se relacionan de alguna manera con el registro actual. Las funciones de ventana son útiles para las tareas de procesamiento, como calcular una media móvil o acceder al valor de las filas en función de la posición relativa de la fila actual.

carga de trabajo

Conjunto de recursos y código que ofrece valor comercial, como una aplicación orientada al cliente o un proceso de backend.

flujo de trabajo

Grupos funcionales de un proyecto de migración que son responsables de un conjunto específico de tareas. Cada flujo de trabajo es independiente, pero respalda a los demás flujos de trabajo del proyecto. Por ejemplo, el flujo de trabajo de la cartera es responsable de priorizar las aplicaciones, planificar las oleadas y recopilar los metadatos de migración. El flujo de trabajo de la cartera entrega estos recursos al flujo de trabajo de migración, que luego migra los servidores y las aplicaciones.

WORM

Consulte escritura única y lectura múltiple.

WQF

Consulte AWS Workload Qualification Framework.

escritura única y lectura múltiple (WORM)

Modelo de almacenamiento que escribe los datos una sola vez y evita que se eliminen o modifiquen. Los usuarios autorizados pueden leer los datos tantas veces como sea necesario, pero no los pueden cambiar. Esta infraestructura de almacenamiento de datos se considera inmutable.

Z

ataque de día cero

Ataque, normalmente de malware, que se aprovecha de una vulnerabilidad de día cero.

vulnerabilidad de día cero

Un defecto o una vulnerabilidad sin mitigación en un sistema de producción. Los agentes de amenazas pueden usar este tipo de vulnerabilidad para atacar el sistema. Los desarrolladores suelen darse cuenta de la vulnerabilidad a raíz del ataque.

peticiones desde cero

Proporcionar a un LLM instrucciones para llevar a cabo una tarea, pero sin ejemplos (pasos) que puedan ayudar a guiarlo. El LLM debe usar los conocimientos del entrenamiento previo para

llevar a cabo la tarea. La eficacia de la petición desde cero depende de la complejidad de la tarea y de la calidad de la petición. Consulte también peticiones con pocos pasos.

aplicación zombi

Aplicación que utiliza un promedio de CPU y memoria menor al 5 por ciento. En un proyecto de migración, es habitual retirar estas aplicaciones.

Las traducciones son generadas a través de traducción automática. En caso de conflicto entre la traducción y la version original de inglés, prevalecerá la version en inglés.