

El futuro está aquí: Descubre los beneficios de los módulos federados

En esta presentación, aprenderemos sobre Module Federation y cómo esta tecnología está cambiando la forma en que desarrollamos aplicaciones basadas en microfrontends.

Fundamentos

¿Qué es Module Federation?

Fundamentos

Los primeros pasos:

- Module Federation se originó en el mundo de Webpack, un popular empaquetador de JavaScript.
- A principios de la década de 2020, los desarrolladores web enfrentaron desafíos al intentar crear aplicaciones front-end más grandes y complejas.

El surgimiento de las arquitecturas de microfrontends:

- A medida que las aplicaciones web crecían, surgió la necesidad de dividirlas en partes más pequeñas y manejables.
- Se popularizaron las arquitecturas de microfrontends, que dividieron las aplicaciones en módulos independientes.



¿Qué es Module Federation?

Fundamentos

- Module Federation es una **tecnología de vanguardia** que está transformando la arquitectura de las aplicaciones frontend.
- Permite a diferentes equipos de desarrollo crear y **mantener microfrontends** de manera independiente, lo que facilita la **colaboración y la escalabilidad**.
- En esencia, Module Federation permite que las partes de una aplicación se desarrollos y se ejecuten de **forma independiente**, pero se integren de **manera eficiente** en una experiencia de usuario unificada.
- Simplificando, es como si cada microfrontend fuera un **módulo** que se puede cargar **dinámicamente en tiempo de ejecución** para construir la aplicación completa.

Beneficios de utilizar Module Federation

Modularidad y flexibilidad

Module Federation permite a los desarrolladores crear aplicaciones web más modulares y flexibles. Esto se debe a que los módulos federados pueden estar ubicados en diferentes orígenes, lo que permite que los desarrolladores combinen diferentes módulos de diferentes fuentes para crear aplicaciones web personalizadas.

Un ejemplo de esto es una aplicación web que tiene un módulo de encabezado y un módulo de pie de página. El módulo de encabezado y el módulo de pie de página podrían estar ubicados en diferentes orígenes, como GitHub o NPM. Esto permitiría a los desarrolladores personalizar los encabezados y pies de página de las páginas sin tener que duplicar el código.

Beneficios de utilizar Module Federation

Reutilización de código

Module Federation facilita la **reutilización de código** entre diferentes aplicaciones web. Esto se debe a que los módulos federados pueden importarse y exportarse fácilmente, lo que permite que los desarrolladores compartan código entre diferentes proyectos.

Un ejemplo de esto es una aplicación web que utiliza un módulo de terceros para proporcionar una funcionalidad específica, como una API de mapas o una API de chat. El módulo de terceros podría estar ubicado en GitHub o NPM. Esto permitiría a los desarrolladores **ahorrar tiempo y esfuerzo** al no tener que desarrollar su propia funcionalidad desde cero.

Beneficios de utilizar Module Federation

Escalabilidad

Module Federation permite escalar aplicaciones web. Esto se debe a que los módulos federados pueden distribuirse entre diferentes servidores, lo que permite que las aplicaciones web puedan manejar más tráfico.

Un ejemplo de esto es una aplicación web que tiene una gran cantidad de tráfico. La aplicación web podría distribuir su lógica entre diferentes servidores utilizando Module Federation. Esto permitiría a la aplicación web manejar más tráfico sin tener que aumentar el tamaño de los servidores individuales.

Conceptos Clave - Host y Remote

Module Federation - Host

- El **host** es la **aplicación principal** que consume módulos compartidos desde otras aplicaciones.
- En la **configuración** de Module Federation, el host define cómo se cargarán los módulos remotos y qué módulos desea utilizar.
- Utiliza una **sintaxis especial** para importar módulos remotos, como `import("remote-app/module")`.
- Puede acceder a los módulos compartidos **expuestos** por las aplicaciones remotas y utilizarlos en su propia aplicación.
- Facilita la creación de aplicaciones modulares y **escalables** al integrar funcionalidades compartidas de manera eficiente.

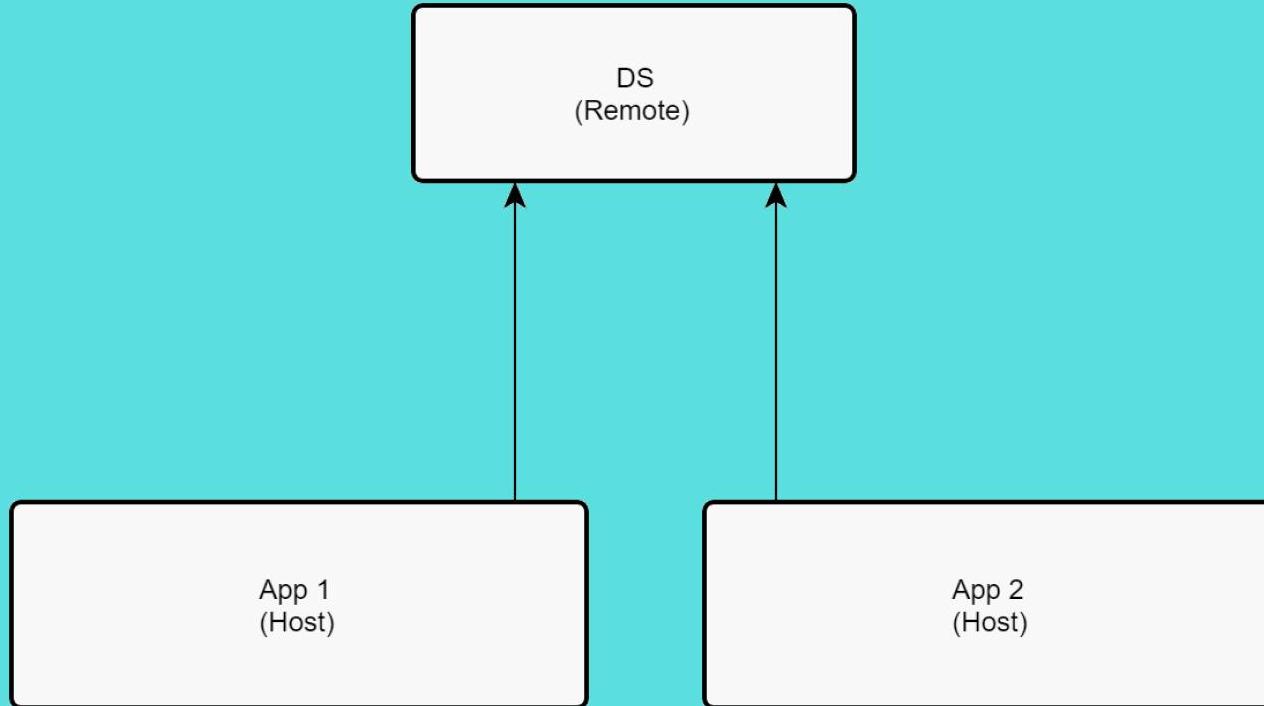
Conceptos Clave - Host y Remote

Module Federation – Remote

- Un **remote** es una aplicación que comparte sus módulos con otras aplicaciones a través de **Module Federation**.
- En la configuración de Module Federation, el remote define qué módulos se expondrán para que otros los utilicen.
- Los remotos permiten la compartición de código y funcionalidades entre aplicaciones independientes.
- Cada remote tiene su propio **punto de entrada** (entry point) y puede **exponer** múltiples módulos.
- Facilita la **creación de microfrontends** y la colaboración entre equipos de desarrollo al compartir componentes y lógica de manera controlada.

Conceptos Clave - Host y Remote

Module Federation - Remote



Comunicación de Datos en Módulos Federados

Introducción

- Module federation permite a los módulos compartir código entre sí.
- Sin embargo, los módulos federados pueden estar ubicados en diferentes orígenes, lo que dificulta la comunicación de datos entre ellos.
- Hay varios métodos que se pueden utilizar para comunicar datos entre módulos federados.

Comunicación de Datos en Módulos Federados

Métodos de comunicación de datos

- **Mensajería:** Los módulos federados pueden intercambiar datos utilizando un servicio de mensajería.
- **API:** Los módulos federados pueden llamar a funciones de otros módulos utilizando una API.
- **Comunicación directa:** Los módulos federados pueden comunicarse entre sí directamente, utilizando un protocolo de comunicación específico.

Comunicación de Datos en Módulos Federados

Ventajas y desventajas de cada método

- **Mensajería:** Es un método flexible y escalable, pero puede ser complejo de implementar.
- **API:** Es un método simple de implementar, pero puede ser menos flexible y escalable que la mensajería.
- **Comunicación directa:** Es el método más eficiente, pero puede ser más complejo de implementar que la mensajería o la API.

Comunicación de Datos en Módulos Federados

Futuro

Middleware

```
new ModuleFederationPlugin({
  middleware: './src/federation-middleware.js'
})

//federation-middleware.js

module.exports = {
  beforeInit,
  beforeLoadingRemote,
  loadRemoteMatch,
  loadRemote,
  errorLoadRemote,
  beforeLoadShare,
  beforePreloadRemote
}
```

Mental Model

Conceptos

- **Remoto:** El remoto es una aplicación construida y desplegada por separado. Puede exponer módulos EcmaScript que pueden cargarse en otras aplicaciones.
- **Anfitrión:** El anfitrión carga uno o varios remotos según sea necesario. Desde la perspectiva de su marco de trabajo, esto se parece a una carga perezosa tradicional. La gran diferencia es que el anfitrión no conoce los remotos en el momento de la compilación.

Mental Model

Conceptos

- **Dependencias Compartidas:** Si varios remotos y el anfitrión utilizan la misma biblioteca, es posible que no desee descargarla varias veces. En su lugar, es posible que desee descargarla una sola vez y compartirla en tiempo de ejecución. Para este caso de uso, el modelo mental permite definir tales dependencias compartidas.
- **Incompatibilidad de Versiones:** Si dos o más aplicaciones utilizan una versión diferente de la misma biblioteca compartida, debemos evitar una incompatibilidad de versiones. Para hacer frente a esto, el modelo mental define varias estrategias, como retroceder a otra versión que se adapte a la aplicación, utilizar una versión compatible diferente (según la versión semántica) o generar un error.

Incompatibilidad de Versiones

Ejemplo

Imagina que tienes dos aplicaciones web, Aplicación A y Aplicación B, que utilizan una biblioteca JavaScript llamada "LibVersion" para realizar algunas operaciones matemáticas.

- Cuando intentas ejecutar **Aplicación A** con la nueva **versión 2.0** de "LibVersion" (la misma que usa Aplicación B), puede que experimentes errores o resultados incorrectos porque la versión 2.0 ha cambiado la forma en que se comporta.
- Por otro lado, si intentas ejecutar **Aplicación B** con la antigua **versión 1.0** de "LibVersion" (la misma que usa Aplicación A), también puedes encontrarte con problemas, ya que la versión 1.0 no tiene las nuevas características que Aplicación B necesita.

Incompatibilidad de Versiones

Ejemplo

- 1. Cargar solo la versión más compatible:** Por defecto, Module Federation carga únicamente la versión más compatible de una dependencia compartida. Por ejemplo, si el anfitrión utiliza la versión 10.0 y un remoto utiliza la versión 10.1, Module Federation podría decidir cargar solo la versión 10.1, ya que debería ser compatible hacia atrás con la versión 10.0.
- 2. Cargar múltiples versiones:** Si dos o más aplicaciones requieren versiones incompatibles, Module Federation carga ambas versiones. Esto ocurre, por ejemplo, cuando el anfitrión hace referencia a la versión 10.0 y el remoto hace referencia a la versión 11.0.
- 3. Evitar múltiples versiones:** Module Federation puede configurarse para tratar una dependencia dividida como un singleton. En este caso, solo se cargará la versión más alta, sin importar si es compatible hacia atrás o no. Si existe el riesgo de una incompatibilidad de versiones, Module Federation emitirá una advertencia en la consola de JavaScript. Si se prefiere, Module Federation también puede generar una excepción en su lugar, de manera que, por ejemplo, las pruebas de integración se den cuenta rápidamente del problema en cuestión.

Tipos versionado de microfrontends

Monorepositorio

Se refiere a las aplicaciones en las que todos los proyectos se encontrarán al final dentro del mismo repositorio.

Aunque esto pudiera parecer algo contradictorio con la teoría de **microfrontends**, donde se buscan generar proyectos que sean independientes, para integrarlos luego, en este caso se tiene para todos los proyectos un **sistema único de dependencias** y subcarpetas para cada uno de estos proyectos.

Un ejemplo de este tipo de microfrontend es el que da por defecto **Angular** para generar internamente sub-aplicaciones.

Tipos versionado de microfrontends

Multirrepositorio

En este tipo se tienen los proyectos en diferentes repositorios. Esto implica tener los proyectos más aislados, con un sistema de **dependencias independiente** para cada proyecto.

Tipos versionado de microfrontends

Metarrepositorio

Se refiere a la estrategia que busca adoptar los dos tipos anteriores. Así, en este tipo se tendrán **múltiples repositorios**, pero además se tendrá uno donde todos ellos se integrarían.

Cada uno de estos tipos tiene sus **ventajas y desventajas**, teniendo cada uno diferentes formas de aminorar sus desventajas.

Es una alternativa a los **SubModules** en Git

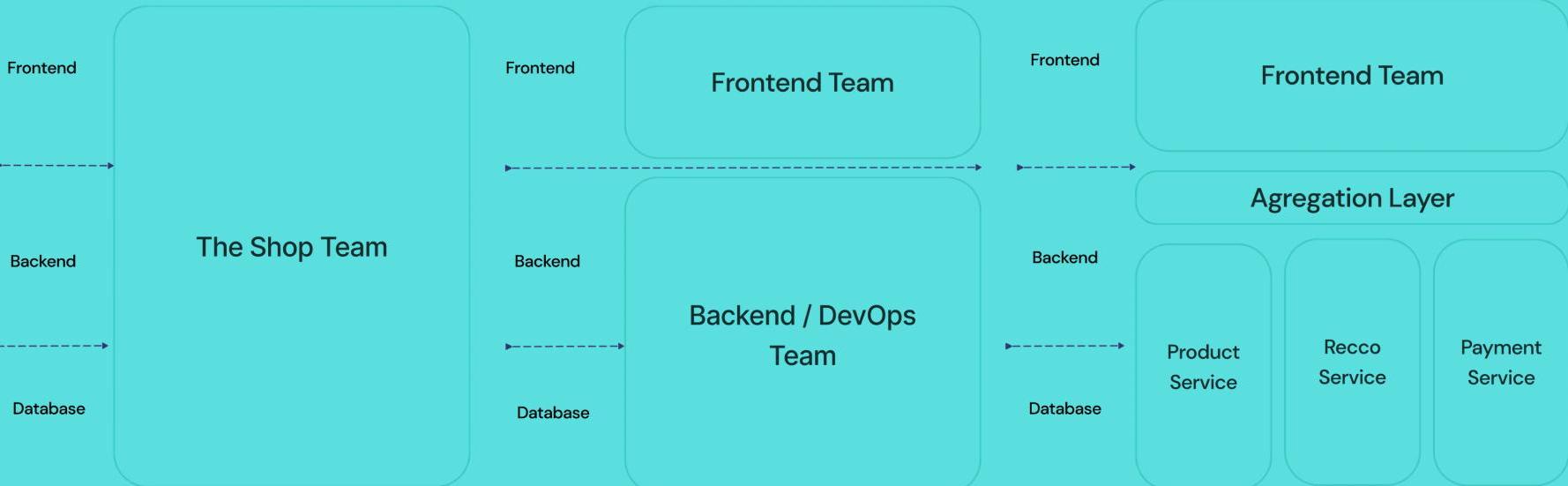
Arquitectura micro-frontend

Frontends monolíticos 1

Monolith

Front and Back

Microservices



Arquitectura micro-frontend

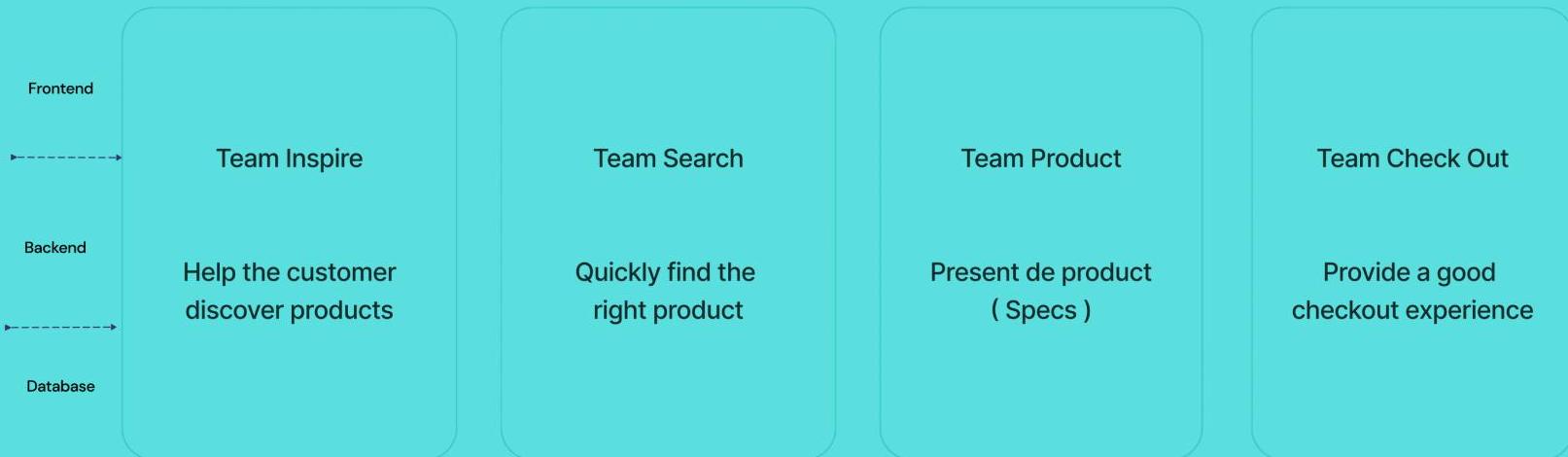
Frontends monolíticos 2

- Demasiado grande y complejo para que alguien lo entienda totalmente y pueda hacer cambios rápida y correctamente.
- Un cambio en el código frontend podría afectar a todo el sitio web.
- Toda modificación en el código frontend debe volverse a implementar, aumentando el tiempo de compilación.

Arquitectura micro-frontend

Organización vertical 1

End-to-End Teams with Micro Frontends



Arquitectura micro-frontend

Organización vertical 2

Así llega la organización vertical, con una arquitectura menos compleja gracias a los microfrontends, que dividen la aplicación en pequeñas funciones independientes, implementando simultáneamente cada una, desde el backend hasta el frontend, por un equipo de trabajo.

Este enfoque se ha hecho popular debido a los problemas que presenta el enfoque monolítico. El frontend ha crecido rápidamente y con una arquitectura monolítica se hace más difícil de mantener.

Como funciona Module Federation?

Parte técnica

- **Federación de módulos:** La federación de módulos implica dividir una aplicación en múltiples partes (módulos) que pueden estar ubicadas en servidores o dominios diferentes. Cada módulo federado se exporta de manera especial utilizando `expose` en la configuración de Webpack.
- **Contenedor:** El módulo principal de la aplicación (el "contenedor") importa estos módulos federados utilizando una función especial proporcionada por Webpack llamada `import()`. Esto permite cargar dinámicamente los módulos federados cuando sea necesario.

Como funciona Module Federation?

Parte técnica

- **Webpack y JSONP:** Cuando el código del contenedor se ejecuta en el navegador y llama a `import()`, Webpack realiza una solicitud HTTP para cargar el módulo federado desde el servidor remoto. Webpack utiliza la técnica JSONP (JSON with Padding) para realizar esta solicitud. JSONP es una técnica que elude la política de "misma origen" al agregar un script dinámico al documento HTML que puede cargar recursos desde dominios diferentes.
- **Cabeceras CORS:** Para que esto funcione sin problemas, el servidor que aloja los módulos federados debe tener configuradas las cabeceras CORS (Cross-Origin Resource Sharing) de manera adecuada. Estas cabeceras permiten a los navegadores descargar recursos desde dominios diferentes al dominio principal de la aplicación, siempre que el servidor remoto lo permita.

Native Federation

Features

Features de Native Federation

- **Mental Model:** Proporciona una forma lógica de pensar sobre cómo funcionan los módulos compartidos.
- **Future Proof:** No depende de herramientas de construcción específicas ni de frameworks.
- **Aprovecha las Nuevas Tecnologías:** Utiliza Import Maps, una tecnología emergente del navegador, y módulos EcmaScript.
- **Configuración Sencilla:** Fácil de configurar y adaptar a tus necesidades.
- **Velocidad Impresionante:** La implementación de referencia utiliza esbuild, que es conocida por su velocidad en la construcción de proyectos. También almacena en caché las dependencias compartidas, lo que acelera aún más el proceso. Además, puedes usarlo con otras herramientas de construcción si lo prefieres.

¿Cuándo usar microfrontends?

Ejemplos

- **Aplicaciones grandes y complejas**
- **Equipos de desarrollo distribuidos**
- **Tecnologías variadas**
- **Escalabilidad**
- **Independencia del ciclo de vida**
- **Pruebas y mantenimiento más sencillos**
- **Aplicaciones con múltiples equipos de diseño**

 Zarzalejo

 ArturoZarzalejo

 ArturoZarzalejo

