

Angular 18: Nuevas Funcionalidades

26 de julio de 2025

3 min de lectura

Por hgaruna



Angular 18: Nuevas Funcionalidades

Angular 18 llega con una serie de mejoras y nuevas características diseñadas para mejorar la productividad del desarrollador, el rendimiento de las aplicaciones y la experiencia del usuario. Esta versión se centra en la optimización, la simplificación del desarrollo y la adopción de las últimas tecnologías. En este artículo, exploraremos las funcionalidades más destacadas de Angular 18.

I Standalone Components: Simplificando el Desarrollo

Una de las mejoras más significativas en Angular 18 es la maduración de los componentes standalone. Esto significa que ahora puedes crear componentes completamente independientes sin necesidad de módulos NgModule. Esta simplificación reduce la complejidad del código y facilita el desarrollo de aplicaciones más pequeñas y modulares.

Ventajas de los Componentes Standalone

- Reducción del código boilerplate.
- Mayor modularidad y facilidad de mantenimiento.
- Simplificación del proceso de aprendizaje para desarrolladores nuevos.
- Mejor organización del código.

Ejemplo de Componente Standalone

```
import { Component } from '@angular/core';

@Component({
  selector: 'app-my-component',
  standalone: true,
  template: `

    Hola desde un componente standalone!

`})
export class MyComponent {}
```

I Mejoras en el Router

El enrutador de Angular también ha recibido mejoras en esta versión. Se ha simplificado la configuración y se ha mejorado la eficiencia. La integración con los componentes standalone es ahora más fluida.

Enrutamiento con Componentes Standalone

La configuración del enrutamiento es ahora más concisa y legible, especialmente al usar componentes standalone:

```
import { Route } from '@angular/router';
import { MyComponent } from './my.component';

const routes: Route[] = [
  { path: '', component: MyComponent },
];
```

I Optimizaciones de Rendimiento

Angular 18 incluye varias optimizaciones que mejoran el rendimiento de las aplicaciones. Estas optimizaciones se centran en la reducción del tamaño del bundle y la mejora de la velocidad de carga.

- Mejoras en el proceso de compilación (AOT).
- Reducción del tamaño del bundle a través de optimizaciones de código.
- Mejoras en la gestión de la memoria.

I Actualizaciones en Angular CLI

La Angular CLI, la herramienta de línea de comandos para crear y gestionar proyectos Angular, también ha recibido actualizaciones en la versión 18. Estas mejoras incluyen un mejor soporte para los componentes standalone y una experiencia de usuario más intuitiva.

Nuevas opciones de la CLI

- Generación de componentes standalone con la opción `--standalone`.
- Mejoras en la generación de scaffolding.
- Mejoras en el reporte de errores.

I TypeScript 5 Support

Angular 18 ahora soporta TypeScript 5, lo que permite a los desarrolladores aprovechar las nuevas características y mejoras de rendimiento de TypeScript. Esto incluye la mejora del sistema de tipos y nuevas funcionalidades del lenguaje.

Ventajas de usar TypeScript 5

- Mejoras en la inferencia de tipos.
- Nuevas características del lenguaje que simplifican el código.
- Mayor seguridad de tipos.

I Conclusión

Angular 18 representa un paso significativo en la evolución de Angular, ofreciendo una serie de mejoras que simplifican el desarrollo, mejoran el rendimiento y optimizan la experiencia del usuario. Las nuevas funcionalidades, como los componentes standalone, las optimizaciones de rendimiento y el soporte para TypeScript 5, hacen de Angular 18 una actualización importante para cualquier desarrollador que trabaje con este framework.

 Compartir en Twitter

 Compartir en Facebook

 Compartir en LinkedIn

 Compartir por WhatsApp

¿Te gustó este artículo?

Suscríbete a nuestro boletín para recibir más contenido como este directamente en tu correo.

Tu correo electrónico

Suscribirme

Ansible: Automatización de Configuración

26 de julio de 2025

3 min de lectura

Por hgaruna

Ansible: Automatización de Configuración

Ansible es una herramienta de automatización de configuración de código abierto, ampliamente utilizada en entornos DevOps y Cloud. Se basa en un enfoque basado en agentes, lo que significa que no requiere la instalación de ningún agente en los nodos administrados. Su simplicidad, potencia y facilidad de uso lo han convertido en una herramienta esencial para la gestión de infraestructuras a escala, permitiendo la automatización de tareas repetitivas y la gestión consistente de la configuración de servidores y aplicaciones.

■ ¿Qué es Ansible y cómo funciona?

Ansible utiliza un lenguaje declarativo simple basado en YAML para definir la configuración deseada. Este lenguaje es legible por humanos y facilita la creación y gestión de playbooks, que son archivos que describen las tareas a realizar en uno o más servidores. Ansible se conecta a los servidores a través de SSH, ejecutando módulos en cada nodo para aplicar la configuración deseada. Este proceso se realiza sin agentes, lo que simplifica la implementación y la gestión.

■ Instalación y Configuración Básica

La instalación de Ansible es relativamente sencilla. En sistemas basados en Linux, generalmente se realiza a través del gestor de paquetes de la distribución. Por ejemplo, en sistemas basados en Debian o Ubuntu, se puede instalar usando:

```
sudo apt update  
sudo apt install ansible
```

Una vez instalado, es necesario configurar el archivo de inventario (`/etc/ansible/hosts`), que define los servidores a gestionar. Un ejemplo simple podría ser:

```
[webservers]  
web1 ansible_host=192.168.1.100  
web2 ansible_host=192.168.1.101
```

I Creación de Playbooks

Los playbooks son el corazón de Ansible. Son archivos YAML que describen las tareas a realizar en los servidores. Un ejemplo simple que instala el paquete Apache en los servidores web definidos en el inventario:

```
---
- hosts: webservers
  become: true
  tasks:
    - name: Install Apache
      apt:
        name: apache2
        state: present
```

En este ejemplo, `hosts: webservers` especifica que las tareas se ejecutarán en los servidores definidos en el grupo `webservers` del inventario. `become: true` permite ejecutar las tareas con privilegios de root. `apt:` es un módulo de Ansible que gestiona los paquetes APT.

I Ventajas y Desventajas de Ansible

Ventajas:

- Fácil de aprender y usar.
- Sin agentes: simplifica la implementación y la gestión.
- Gran comunidad y soporte.
- Amplia gama de módulos para diversas tareas.
- Idempotente: las tareas se pueden ejecutar varias veces sin efectos secundarios.

Desventajas:

- Puede ser menos eficiente que otras herramientas para tareas complejas.
- La gestión de estados complejos puede requerir playbooks más elaborados.
- Depende de SSH, lo que puede ser un cuello de botella en entornos con problemas de red.

I Casos de Uso y Ejemplos Avanzados

Ansible se puede utilizar para una amplia gama de tareas, incluyendo:

- Configuración de servidores web (Apache, Nginx).
- Gestión de bases de datos (MySQL, PostgreSQL)

- Implementación de aplicaciones.
- Automatización de la gestión de redes.
- Orquestación de la nube (AWS, Azure, GCP).

Para ejemplos más avanzados, se pueden explorar módulos para la gestión de configuraciones de red, la implementación de contenedores Docker o la gestión de servicios de nube.

I Conclusión

Ansible es una herramienta poderosa y versátil para la automatización de configuración en entornos DevOps y Cloud. Su simplicidad, eficiencia y gran comunidad lo convierten en una opción ideal para equipos que buscan mejorar la eficiencia y la consistencia en la gestión de sus infraestructuras. A pesar de algunas limitaciones, la facilidad de uso y la amplia gama de funcionalidades hacen que Ansible sea una herramienta invaluable para cualquier profesional de DevOps.

 Compartir en Twitter

 Compartir en Facebook

 Compartir en LinkedIn

 Compartir por WhatsApp



hgaruna

  

¿Te gustó este artículo?

Suscríbete a nuestro boletín para recibir más contenido como este directamente en tu correo.

Tu correo electrónico

Suscríbeme

Backup y recuperación de datos

2025-07-28

5 min de lectura

Por hgaruna

La gestión de copias de seguridad y la recuperación de datos son aspectos cruciales en el desarrollo y mantenimiento de cualquier aplicación que utilice bases de datos. Una estrategia sólida de backup y recuperación garantiza la continuidad del negocio y protege contra la pérdida de información valiosa, ya sea por fallos del hardware, errores humanos o ciberataques. Este artículo explorará las mejores prácticas para implementar un sistema robusto de backup y recuperación de datos para bases de datos, cubriendo diferentes estrategias y tecnologías.

I Estrategias de Copias de Seguridad

Existen diversas estrategias para realizar copias de seguridad de bases de datos, cada una con sus propias ventajas y desventajas. La elección de la estrategia óptima dependerá de factores como el tamaño de la base de datos, la frecuencia de actualizaciones, los requisitos de recuperación y los recursos disponibles.

Copias de Seguridad Completas

Una copia de seguridad completa, también conocida como full backup, crea una copia completa de la base de datos en un momento específico. Este método es simple y proporciona un punto de restauración completo, pero puede ser lento y requerir un gran espacio de almacenamiento, especialmente para bases de datos grandes. Se recomienda realizar copias de seguridad completas con regularidad, por ejemplo, semanalmente o mensualmente.

Copias de Seguridad Incrementales

Las copias de seguridad incrementales (incremental backups) solo copian los datos que han cambiado desde la última copia de seguridad completa o incremental. Esto reduce significativamente el tiempo y el espacio de almacenamiento necesarios en comparación con las copias de seguridad completas. Sin embargo, para restaurar la base de datos, se necesita la última copia de seguridad completa y todas las copias de seguridad incrementales posteriores.

Copias de Seguridad Diferenciales

Las copias de seguridad diferenciales (differential backups) copian todos los datos que han cambiado desde la última copia de seguridad completa. A diferencia de las incrementales, cada copia diferencial contiene todos los cambios desde la última copia completa, lo que simplifica el proceso de restauración. Sin embargo, las copias diferenciales suelen ser más

grandes que las incrementales.

Las copias de seguridad incrementales solo copian los datos que se han modificado desde la última copia completa. Esto hace que las copias incrementales sean más rápidas y eficientes que las completas.

I | Tecnologías de Copias de Seguridad

Existen diversas herramientas y tecnologías para realizar copias de seguridad de bases de datos. Algunas son específicas para un tipo de base de datos, mientras que otras son más generales.

Herramientas de la propia base de datos

Muchas bases de datos, como MySQL, PostgreSQL y SQL Server, ofrecen sus propias utilidades para realizar copias de seguridad. Estas herramientas suelen ser eficientes y están integradas con la base de datos, lo que facilita su uso.

```
-- Ejemplo de copia de seguridad en MySQL  
mysqldump -u usuario -p base_de_datos > backup.sql
```

Herramientas de terceros

Existen numerosas herramientas de terceros que proporcionan funcionalidades avanzadas para la gestión de copias de seguridad, incluyendo la programación de backups, la compresión de datos y la encriptación. Ejemplos de estas herramientas incluyen Bacula, Amanda y rsync.

Almacenamiento en la nube

El almacenamiento en la nube ofrece una solución segura y escalable para almacenar copias de seguridad de bases de datos. Servicios como Amazon S3, Google Cloud Storage y Azure Blob Storage permiten almacenar grandes cantidades de datos con alta disponibilidad y redundancia.

I | Recuperación de Datos

La recuperación de datos es el proceso de restaurar una base de datos a un estado anterior a partir de una copia de seguridad. La velocidad y la facilidad de la recuperación dependen de la estrategia de copia de seguridad implementada y de la calidad de las copias de seguridad.

Proceso de Recuperación

1. **Identificar el punto de restauración:** Determinar la copia de seguridad más adecuada para restaurar la base de datos.
2. **Restaurar la copia de seguridad:** Utilizar las herramientas apropiadas para restaurar la copia de seguridad a un servidor o instancia de base de datos.
3. **Verificar la integridad de los datos:** Despues de la restauración, verificar la integridad y la consistencia de los datos.
4. **Documentar el proceso:** Registrar todos los pasos del proceso de recuperación para futuras referencias.

Ejemplos de Recuperación

El proceso de restauración varía según la herramienta y el tipo de base de datos. A continuación, se muestra un ejemplo básico de restauración desde una copia de seguridad SQL:

```
-- Ejemplo de restauración en MySQL  
mysql -u usuario -p base_de_datos < backup.sql
```

I Mejores Prácticas

- **Prueba regularmente tus copias de seguridad:** Realiza pruebas de restauración periódicas para asegurar que tus copias de seguridad son recuperables.
- **Almacenamiento fuera de sitio:** Almacena tus copias de seguridad en una ubicación diferente a la ubicación principal de la base de datos para protegerte contra desastres locales.
- **Encriptación de datos:** Encripta tus copias de seguridad para proteger la información confidencial.
- **Automatización:** Automatiza el proceso de copia de seguridad para garantizar la regularidad y la consistencia.
- **Retención de datos:** Define una política de retención de datos para determinar cuánto tiempo se deben conservar las copias de seguridad.

Implementar una estrategia sólida de backup y recuperación de datos es fundamental para la continuidad del negocio y la protección de la información valiosa. La elección de la estrategia y las herramientas adecuadas dependerá de las necesidades específicas de cada organización, pero la planificación y la prueba regular son cruciales para el éxito.

#Seguridad

#IA

#Bases de Datos

Compartir este artículo:

 Twitter

 Facebook

 LinkedIn

 WhatsApp



hgaruna

[Twitter](#) [Q](#) [in](#)

¿Te gustó este artículo?

Suscríbete a nuestro boletín para recibir más contenido como este directamente en tu correo.

Tu correo electrónico

Tu correo electrónico

[Suscribirme](#)

CDN: Content Delivery Networks

2025-07-29

4 min de lectura

Por hgaruna

En el mundo del desarrollo web, la velocidad de carga es crucial para la experiencia del usuario y el posicionamiento SEO. Una de las estrategias más efectivas para mejorar el rendimiento de un sitio web es la implementación de una Red de Distribución de Contenido (CDN, por sus siglas en inglés). Este artículo explorará en detalle qué son las CDN, cómo funcionan y cómo pueden beneficiar a tu sitio web.

I. ¿Qué es una CDN?

Una CDN es una red global de servidores distribuidos geográficamente que almacenan copias de los recursos de tu sitio web, como imágenes, hojas de estilo (CSS), archivos JavaScript y otros archivos estáticos. Cuando un usuario accede a tu sitio, la solicitud se enruta al servidor CDN más cercano a su ubicación geográfica. Esto reduce significativamente el tiempo de carga, ya que la información viaja una distancia menor.

Beneficios Clave de las CDN

Las CDN ofrecen una variedad de beneficios, incluyendo:

- **Mayor velocidad de carga:** Reducción significativa del tiempo de carga para los usuarios.
- **Mejor experiencia del usuario:** Sitios web más rápidos y receptivos conducen a una mejor experiencia.
- **Aumento del tráfico:** Una mejor experiencia del usuario generalmente resulta en un mayor tráfico.
- **Mayor escalabilidad:** Las CDN pueden manejar picos de tráfico sin afectar el rendimiento.
- **Reducción de la carga en el servidor principal:** Las CDN alivian la carga del servidor principal al manejar las solicitudes de archivos estáticos.
- **Mejor SEO:** La velocidad de carga es un factor de clasificación importante para los motores de búsqueda.

I. Cómo funcionan las CDN

Una CDN funciona mediante una serie de pasos:

1. **Solicitud del usuario:** Un usuario accede a tu sitio web.
2. **Enrutamiento inteligente:** El sistema CDN determina el servidor más cercano al usuario.
3. **Entrega de contenido:** El servidor CDN entrega el contenido solicitado al usuario.
4. **Cacheo:** El servidor CDN almacena en caché una copia del contenido para futuras solicitudes.
5. **Actualización del caché:** El sistema CDN se actualiza periódicamente para asegurar que el contenido sea el más reciente.

Ejemplo de integración con un CDN (Cloudflare)

La integración con un CDN como Cloudflare suele ser sencilla. Normalmente implica apuntar tus registros DNS a los servidores de Cloudflare. Después, Cloudflare se encargará de la distribución de tu contenido.

```
// Ejemplo de cómo se podría implementar una llamada a una API para obtener información de la
// CDN (ejemplo conceptual)
fetch('https://api.cloudflare.com/client/v4/zones//dns_records', {
  method: 'GET',
  headers: {
    'X-Auth-Email': '',
    'X-Auth-Key': '',
  },
})
.then(response => response.json())
.then(data => {
  // Procesar la respuesta de la API
  console.log(data);
})
.catch(error => {
  console.error('Error:', error);
});
```

I Tipos de CDN

Existen diferentes tipos de CDN, cada uno con sus propias características:

- **CDN de almacenamiento en caché:** Este tipo de CDN se centra principalmente en almacenar en caché archivos estáticos.
- **CDN de streaming de vídeo:** Especializadas en la entrega de contenido de vídeo.
- **CDN de juegos:** Optimizadas para la entrega de contenido de juegos en línea.
- **CDN de borde:** Procesan solicitudes cerca de los usuarios para una mayor eficiencia.

I Consideraciones al elegir una CDN

Al elegir una CDN, debes considerar factores como:

- **Precio:** El costo de la CDN puede variar significativamente.
- **Rendimiento:** La velocidad y la fiabilidad de la CDN son cruciales.
- **Características:** Busca características adicionales como seguridad, optimización de imágenes y soporte para diferentes protocolos.
- **Integración:** Asegúrate de que la CDN se integre fácilmente con tu sitio web y otros servicios.

Si te gustó este artículo, te recomendamos que leas estos otros artículos de la misma categoría:

[¿Qué es un CDN y cómo puede mejorar el rendimiento de tu sitio web?](#)

[¿Qué es una API y cómo puedes usarla para mejorar tu sitio web?](#)

Conclusión

Las CDN son una herramienta esencial para cualquier sitio web que busque mejorar su rendimiento y escalabilidad. Al comprender cómo funcionan las CDN y al elegir la opción adecuada para tus necesidades, puedes mejorar significativamente la experiencia de tus usuarios y el éxito de tu sitio web.

#JavaScript

#API

#Seguridad

#Performance

#IA

Compartir este artículo:

 Twitter

 Facebook

 LinkedIn

 WhatsApp



hgaruna

  

¿Te gustó este artículo?

Suscríbete a nuestro boletín para recibir más contenido como este directamente en tu correo.

Tu correo electrónico

Tu correo electrónico

Suscribirme

Chatbots: Implementación práctica

2025-07-28

4 min de lectura

Por hgaruna

Los chatbots se han convertido en una herramienta esencial para las empresas que buscan automatizar interacciones con clientes y mejorar la eficiencia operativa. Este artículo profundiza en la implementación práctica de chatbots, cubriendo desde la selección de la plataforma adecuada hasta la consideración de aspectos éticos y de mantenimiento.

I | Seleccionando la Plataforma Adecuada

La elección de la plataforma correcta es crucial para el éxito de tu chatbot. Debes considerar factores como la facilidad de uso, la escalabilidad, las integraciones con otras herramientas y el costo. Existen diversas opciones, desde plataformas de código abierto como Rasa hasta servicios en la nube como Dialogflow de Google y Amazon Lex.

Plataformas de Código Abierto vs. Servicios en la Nube

Las plataformas de código abierto ofrecen mayor flexibilidad y control, pero requieren conocimientos de programación y mantenimiento más exhaustivos. Los servicios en la nube son más fáciles de implementar y gestionar, pero pueden tener limitaciones en cuanto a personalización y costos a largo plazo.

- **Plataformas de Código Abierto (ej. Rasa):** Mayor control, personalización y escalabilidad. Requiere experiencia en desarrollo.
- **Servicios en la Nube (ej. Dialogflow, Amazon Lex):** Fácil implementación, integración con otros servicios de Google/Amazon. Potencialmente más costoso a gran escala.

I | Diseño del Flujo de Conversación

El diseño del flujo de conversación es fundamental para la experiencia del usuario. Un buen diseño debe ser intuitivo, eficiente y capaz de manejar una variedad de escenarios. Se recomienda utilizar diagramas de flujo para visualizar el proceso y asegurar una navegación fluida.

Entidades y Intentos

Para comprender las intenciones del usuario, los chatbots utilizan entidades (información específica, como nombres, fechas o ubicaciones) e intentos (la acción que el usuario quiere realizar). Un buen diseño considera la variedad de formas en que un usuario puede expresar la misma intención.

```
{  
    "intent": "ObtenerInformaciónProducto",  
    "parameters": {  
        "producto": "Zapatillas Nike",  
        "color": "Blanco"  
    }  
}
```

I | Integración con Sistemas Existentes

Para maximizar el impacto, los chatbots deben integrarse con los sistemas existentes de la empresa, como bases de datos de clientes, sistemas de CRM o plataformas de comercio electrónico. Esto permite al chatbot acceder a información relevante y realizar acciones en tiempo real.

Ejemplo de Integración con una Base de Datos

El chatbot puede usar una API para consultar una base de datos y obtener información sobre un producto específico, como su precio o disponibilidad, antes de responder al usuario.

```
# Ejemplo de consulta a una base de datos (Python)  
import sqlite3  
  
conn = sqlite3.connect('productos.db')  
cursor = conn.cursor()  
  
cursor.execute("SELECT precio FROM productos WHERE nombre = ?", ("Zapatillas Nike",))  
precio = cursor.fetchone()[0]  
  
conn.close()  
print(f"El precio de las Zapatillas Nike es: {precio}")
```

I | Pruebas y Optimización

Las pruebas exhaustivas son cruciales para asegurar que el chatbot funcione correctamente y proporcione una experiencia de usuario positiva. Esto incluye pruebas de funcionalidad, pruebas de usuario y análisis de los datos de conversación para identificar áreas de mejora.

Métricas Clave

Algunas métricas clave para monitorizar el rendimiento del chatbot incluyen la tasa de éxito de las conversaciones, el tiempo de respuesta y la satisfacción del usuario.

1. **Tasa de éxito de las conversaciones:** Porcentaje de conversaciones que se completan con éxito.
2. **Tiempo de respuesta:** Tiempo que tarda el chatbot en responder a una consulta.
3. **Satisfacción del usuario:** Medida de la satisfacción del usuario con la interacción con el chatbot.

I Consideraciones Éticas y de Privacidad

Es importante considerar las implicaciones éticas y de privacidad al implementar un chatbot. Se debe asegurar que el chatbot se utilice de manera responsable y que se proteja la privacidad de los usuarios. Esto incluye el cumplimiento de las regulaciones de protección de datos, como el GDPR.

Al implementar un chatbot, es crucial considerar las implicaciones éticas y de privacidad. Se deben establecer reglas claras para el uso del chatbot y garantizar que se respeten los derechos de los usuarios.

I Mantenimiento y Actualizaciones

Los chatbots requieren un mantenimiento continuo para asegurar que sigan funcionando correctamente y que se adapten a las necesidades cambiantes de los usuarios. Esto incluye la actualización del modelo de lenguaje, la adición de nuevas funcionalidades y la corrección de errores.

I Conclusión

La implementación exitosa de un chatbot requiere una planificación cuidadosa, una selección adecuada de la plataforma, un diseño de conversación intuitivo y un enfoque en las pruebas y la optimización continua. Al considerar los aspectos éticos y de privacidad, y al comprometerse con el mantenimiento continuo, las empresas pueden aprovechar al máximo el potencial de los chatbots para mejorar la eficiencia y la experiencia del cliente.

#JavaScript

#Python

#API

#Performance

#IA

#Bases de Datos

Compartir este artículo:

Twitter

Facebook

LinkedIn

WhatsApp



¿Te gustó este artículo?

Suscríbete a nuestro boletín para recibir más contenido como este directamente en tu correo.

Tu correo electrónico

Tu correo electrónico

Suscribirme

CI/CD: Automatización de Despliegues

 26 de julio de 2025

 4 min de lectura

 Por hgaruna

CI/CD: Automatización de Despliegues

En el dinámico mundo del desarrollo de software, la entrega rápida y confiable de aplicaciones es crucial. CI/CD (Integración Continua/Entrega Continua) es una práctica fundamental de DevOps que automatiza el proceso de desarrollo, pruebas y despliegue, permitiendo a los equipos lanzar actualizaciones de software con mayor frecuencia y menor riesgo. Este artículo explorará los componentes clave de CI/CD, sus beneficios, desafíos y cómo implementar una estrategia efectiva para automatizar sus despliegues.

I. ¿Qué es CI/CD?

CI/CD es un conjunto de prácticas y herramientas que automatizan el proceso de desarrollo de software, desde la integración de código hasta la entrega a producción. La **Integración Continua (CI)** se centra en la automatización de la integración de código, ejecutando pruebas automáticamente cada vez que un desarrollador realiza un commit. La **Entrega Continua (CD)** extiende este proceso automatizando el despliegue de la aplicación en diferentes entornos, como pruebas, staging y producción.

A menudo, CD se extiende a **Despliegue Continuo (CD)**, donde cada cambio de código que pasa las pruebas se despliega automáticamente a producción. Es importante notar que mientras CD automatiza el proceso de despliegue, el despliegue continuo automatiza el acto de despliegue mismo.

I. Beneficios de Implementar CI/CD

Adoptar una estrategia CI/CD ofrece numerosos beneficios:

- **Mayor velocidad de entrega:** Automatizar el proceso de despliegue reduce significativamente el tiempo necesario para lanzar nuevas funcionalidades.
- **Reducción de errores:** Las pruebas automatizadas detectan errores tempranamente, reduciendo el costo y el tiempo de corrección.
- **Mejor colaboración en equipo:** CI/CD fomenta la colaboración y la transparencia entre los equipos de desarrollo y operaciones.
- **Mayor frecuencia de lanzamientos:** Permite realizar lanzamientos más pequeños y frecuentes, lo que facilita la gestión de cambios y la respuesta a los comentarios de los usuarios.
- **Mayor calidad del software:** La integración y las pruebas continuas mejoran la calidad general del software.

I Componentes Clave de un Pipeline CI/CD

Un pipeline CI/CD típico incluye las siguientes etapas:

1. **1. Integración Continua:** El código se integra en un repositorio central (como Git) varias veces al día. Se ejecutan pruebas unitarias y de integración automáticamente.
2. **2. Construcción:** El código se compila y se empaqueta en un artefacto desplegable (e.g., un archivo JAR, un contenedor Docker).
3. **3. Pruebas:** Se ejecutan pruebas automatizadas (unitarias, de integración, funcionales, de rendimiento) para validar la calidad del software.
4. **4. Entrega Continua:** El artefacto se despliega en un entorno de pruebas (staging) para realizar pruebas adicionales antes de la producción.
5. **5. Despliegue Continuo (opcional):** El artefacto se despliega automáticamente en producción una vez que se aprueban todas las pruebas.

I Herramientas para la Implementación de CI/CD

Existen numerosas herramientas que facilitan la implementación de CI/CD. Algunas de las más populares incluyen:

- **Jenkins:** Una herramienta de código abierto muy popular y versátil para la automatización de la integración y el despliegue continuo.
- **GitHub Actions:** Una solución integrada en GitHub para automatizar los workflows de CI/CD.
- **GitLab CI/CD:** Similar a GitHub Actions, pero integrado en GitLab.
- **CircleCI:** Una plataforma en la nube para la automatización de CI/CD.
- **Azure DevOps:** Una solución completa de Microsoft para la gestión de proyectos y la automatización de CI/CD.

Ejemplo de un Pipeline de Jenkins con un script de despliegue simple

Configuración de Jenkins

Se asume que ya se tiene un proyecto en un repositorio Git y un servidor Jenkins configurado. Un job de Jenkins podría ser configurado para ejecutar los siguientes comandos:

```
#!/bin/bash

# Clonar el repositorio
git clone https://github.com/usuario/repositorio.git

# Navegar al directorio del proyecto
cd repositorio
```

cd repositorio

```
# Construir la aplicación (ejemplo con Maven)
mvn clean package

# Copiar el archivo JAR al servidor de aplicaciones (ejemplo)
scp target/mi-aplicacion.jar usuario@servidor:/ruta/de/despliegue
```

Script de Despliegue en el Servidor

En el servidor, se necesitaría un script para detener la aplicación existente, copiar el nuevo JAR y reiniciar la aplicación. Ejemplo simple (requiere ajustes según el servidor de aplicaciones):

```
#!/bin/bash

# Detener la aplicación (ejemplo con un script de inicio/detención)
./stop.sh

# Copiar el nuevo JAR
cp /ruta/de/despliegue/mi-aplicacion.jar /ruta/de/aplicacion/

# Iniciar la aplicación (ejemplo con un script de inicio/detención)
./start.sh
```

I Conclusión

La implementación de CI/CD es una inversión significativa pero que ofrece un retorno considerable en términos de velocidad, calidad y eficiencia en el desarrollo de software. Aunque la configuración inicial puede requerir tiempo y esfuerzo, la automatización de los procesos de despliegue proporciona beneficios a largo plazo, permitiendo a los equipos de desarrollo concentrarse en la creación de valor y la innovación.

 Compartir en Twitter

 Compartir en Facebook

 Compartir en LinkedIn

 Compartir por WhatsApp

¿Te gustó este artículo?

Suscríbete a nuestro boletín para recibir más contenido como este directamente en tu correo.

Tu correo electrónico

Suscribirme

Introducción

📅 {{PUBLICATION_DATE}}

⌚ 4 min de lectura

👤 Por hgaruna

I Introducción

La Clean Architecture, popularizada por Robert C. Martin (Uncle Bob), es un enfoque para diseñar software que prioriza la independencia de las capas y la separación de preocupaciones. Este enfoque se alinea perfectamente con los principios SOLID, un conjunto de cinco principios de diseño de objetos que promueven la flexibilidad, la mantenibilidad y la extensibilidad del código. En este artículo, exploraremos la intersección de la Clean Architecture y los principios SOLID, mostrando cómo ambos trabajan juntos para crear sistemas robustos y fáciles de mantener.

I Sección Principal

La Clean Architecture se caracteriza por su estructura en capas concéntricas. El núcleo contiene la lógica de negocio independiente de cualquier framework, base de datos o interfaz de usuario. Las capas externas se encargan de la interacción con el mundo exterior, como la presentación (UI), la infraestructura (bases de datos, servicios externos) y los frameworks. Esta separación de preocupaciones permite que el núcleo permanezca inmutable ante cambios en las capas externas.

Principios SOLID y Clean Architecture

Los principios SOLID son fundamentales para implementar una Clean Architecture efectiva. Veamos cómo cada principio contribuye:

- 1. Principio de Responsabilidad Única (SRP):** Cada clase o módulo debe tener una única razón para cambiar. En la Clean Architecture, esto se refleja en la separación de capas. La capa de dominio (núcleo) se encarga únicamente de la lógica de negocio, mientras que las capas externas manejan la presentación, la persistencia de datos, etc. Cada capa tiene una responsabilidad única y bien definida.
- 2. Principio Abierto/Cerrado (OCP):** Las entidades de software (clases, módulos, funciones) deben estar abiertas para la extensión, pero cerradas para la modificación. La Clean Architecture facilita esto mediante la abstracción. Las interfaces definen contratos que las capas externas implementan, permitiendo agregar nuevas funcionalidades sin modificar el núcleo.
- 3. Principio de Sustitución de Liskov (LSP):** Los subtipos deben ser sustituibles por sus tipos base sin alterar la corrección del programa. En la Clean Architecture, esto se aplica al diseño de interfaces y clases. Si una clase implementa una interfaz, debe cumplir con el contrato definido por dicha interfaz sin romper la funcionalidad.
- 4. Principio de Segregación de Interfaces (ISP):** Las clases no deben depender de métodos que no usan. En la Clean Architecture, esto se traduce en interfaces pequeñas y específicas. En lugar de una gran interfaz que hace muchas cosas, es mejor tener varias interfaces más pequeñas, cada

una con una responsabilidad específica.

5. **Principio de Inversión de Dependencias (DIP):** Las dependencias altas deben depender de abstracciones, no de concreciones. Las abstracciones no deben depender de detalles. Los detalles deben depender de abstracciones. La Clean Architecture aplica este principio al máximo. El núcleo no depende de las capas externas, sino de abstracciones (interfaces). Las capas externas implementan estas interfaces, proporcionando concreciones.

Ejemplo de Código (JavaScript - Capa de Dominio)

Este ejemplo muestra una clase de dominio simple que calcula el precio total de un pedido, siguiendo el principio de responsabilidad única:

```
class Order {  
  constructor(items) {  
    this.items = items;  
  }  
  
  getTotalPrice() {  
    return this.items.reduce((total, item) => total + item.price, 0);  
  }  
}
```

Ejemplo de Código (JavaScript - Capa de Presentación)

Este ejemplo muestra una función que interactúa con la capa de dominio para mostrar el precio total al usuario. Observa cómo depende de una abstracción (una interfaz, en este caso simulada):

```
// Simulación de una interfaz para la capa de dominio  
const OrderService = {  
  calculateTotalPrice: (items) => {  
    const order = new Order(items);  
    return order.getTotalPrice();  
  }  
};  
  
function displayTotalPrice(items) {  
  const totalPrice = OrderService.calculateTotalPrice(items);  
  console.log("Total price:", totalPrice);  
}  
  
const items = [{ price: 10 }, { price: 20 }];  
displayTotalPrice(items);
```

La Clean Architecture, en conjunto con los principios SOLID, proporciona una estructura sólida y escalable para el desarrollo de software. La separación de preocupaciones, la abstracción y la dependencia de interfaces permiten crear sistemas más mantenibles, testables y resistentes a los cambios. Al aplicar estos principios, los desarrolladores pueden construir aplicaciones de alta calidad que se adaptan a las necesidades cambiantes del negocio. Recuerda que la clave está en la disciplina y la comprensión profunda de los principios para lograr una arquitectura limpia y eficiente.

[{{#TAGS}}](#) [#{{.}}](#) [{{/TAGS}}](#)

Compartir este artículo:

[Twitter](#)

[Facebook](#)

[LinkedIn](#)

[WhatsApp](#)



hgaruna

[Twitter](#) [Q](#) [in](#)

¿Te gustó este artículo?

Suscríbete a nuestro boletín para recibir más contenido como este directamente en tu correo.

Tu correo electrónico

Suscribirme

Introducción

2025-07-27

5 min de lectura

Por hgaruna

I Introducción

La cobertura de código es una métrica crucial en el desarrollo de software que indica la proporción del código fuente que se ha ejecutado durante las pruebas. Una alta cobertura de código no garantiza la ausencia de errores, pero sí proporciona una fuerte indicación de la exhaustividad de las pruebas y la calidad general del software. Este artículo profundiza en la importancia de la cobertura de código como métrica de calidad, explorando sus diferentes tipos, herramientas de medición y las mejores prácticas para su interpretación y mejora.

I Sección Principal

La cobertura de código se mide generalmente como un porcentaje. Un 100% de cobertura significa que cada línea de código se ha ejecutado al menos una vez durante las pruebas. Sin embargo, alcanzar el 100% no siempre es el objetivo, ni siquiera deseable. La calidad de las pruebas es más importante que el porcentaje de cobertura. Es posible tener una alta cobertura con pruebas de baja calidad que no detecten errores importantes. Por lo tanto, la cobertura de código debe considerarse una métrica complementaria, no la única métrica para evaluar la calidad del software.

Tipos de Cobertura de Código

Existen varios tipos de cobertura de código, cada uno ofreciendo una perspectiva diferente sobre la exhaustividad de las pruebas:

- **Cobertura de líneas (Line Coverage):** Indica el porcentaje de líneas de código que se han ejecutado. Es la métrica más común y fácil de entender.
- **Cobertura de ramas (Branch Coverage):** Mide el porcentaje de ramas de ejecución (como las condiciones `if`, `else`, `switch`) que se han probado. Es más exhaustiva que la cobertura de líneas, ya que identifica posibles caminos de ejecución no probados.
- **Cobertura de condiciones (Condition Coverage):** Va un paso más allá de la cobertura de ramas, analizando cada condición individual dentro de una rama. Por ejemplo, en una condición `if (a > 5 && b < 10)`, la cobertura de condiciones verifica que se hayan probado los casos donde `a > 5` es verdadero y falso, y lo mismo para `b < 10`, independientemente de la combinación.
- **Cobertura de caminos (Path Coverage):** Es el tipo de cobertura más exhaustivo, pero también el más difícil de lograr. Se centra en probar cada posible camino de ejecución a través del código. Para programas complejos, el número de caminos puede ser exponencial, haciendo que la cobertura de caminos sea prácticamente inalcanzable.
- **Cobertura de funciones/métodos (Function/Method Coverage):** Indica el porcentaje de funciones o métodos que se han llamado durante las pruebas.

Herramientas para medir la cobertura de código

Existen numerosas herramientas para medir la cobertura de código, tanto de código abierto como comerciales. La elección de la herramienta dependerá del lenguaje de programación y del entorno de desarrollo. Algunos ejemplos incluyen:

- **Jest (JavaScript)**: Una herramienta de testing popular para JavaScript que proporciona métricas de cobertura de código de forma integrada.
- **pytest-cov (Python)**: Una extensión para pytest que permite medir la cobertura de código en Python.
- **JaCoCo (Java)**: Una herramienta de cobertura de código para Java muy utilizada.
- **SonarQube**: Una plataforma de análisis de código que incluye la medición de la cobertura de código entre otras muchas métricas.

Ejemplo de Cobertura de Código con Jest (JavaScript)

Supongamos que tenemos la siguiente función en JavaScript:

```
function suma(a, b) {  
  if (a > 0 && b > 0) {  
    return a + b;  
  } else {  
    return 0;  
  }  
}
```

Un test con Jest que busca una alta cobertura podría ser:

```
test('Suma dos números positivos', () => {  
  expect(suma(5, 3)).toBe(8);  
});  
  
test('Suma con un número negativo', () => {  
  expect(suma(-5, 3)).toBe(0);  
});  
  
test('Suma con dos números negativos', () => {  
  expect(suma(-5, -3)).toBe(0);  
});  
  
test('Suma con cero', () => {  
  expect(suma(0, 3)).toBe(0);  
});
```

Ejecutar este test con Jest generará un reporte de cobertura, mostrando el porcentaje de líneas, ramas y condiciones cubiertas. Este ejemplo demuestra la importancia de las pruebas para alcanzar una alta cobertura y la necesidad de pruebas que cubran diferentes escenarios.

Interpretando la Cobertura de Código

Un alto porcentaje de cobertura no garantiza un software sin errores, pero una baja cobertura sugiere la posibilidad de errores ocultos. La interpretación de la cobertura de código debe ser contextualizada. Se debe prestar atención a las áreas con baja cobertura, investigando la razón de esta baja cobertura. ¿Son partes del código complejas que requieren más pruebas? ¿Son partes del código que rara vez se utilizan y por lo tanto tienen menor riesgo? Una baja cobertura en áreas críticas del software es mucho más preocupante que una baja cobertura en áreas menos importantes.

I Conclusión

La cobertura de código es una métrica valiosa para evaluar la calidad del software, pero debe utilizarse con prudencia. No es una medida definitiva de la calidad, sino una herramienta que ayuda a identificar áreas que necesitan más atención en las pruebas. Combinar la cobertura de código con otras métricas de calidad, como la revisión de código y las pruebas manuales, proporciona una visión más completa de la calidad del software. El objetivo no debe ser alcanzar un porcentaje arbitrario de cobertura, sino asegurar que las partes críticas del software estén adecuadamente probadas y que se minimice el riesgo de errores.

#JavaScript

#Python

#IA

#Testing

Compartir este artículo:

Twitter

Facebook

LinkedIn

WhatsApp



hgaruna

¿Te gustó este artículo?

Suscríbete a nuestro boletín para recibir más contenido como este directamente en tu correo.

Tu correo electrónico

Tu correo electrónico

Suscribirme

Code Splitting: División de Bundles

26 de julio de 2025

3 min de lectura

Por hgaruna

En el desarrollo web moderno, la optimización del rendimiento es crucial para una experiencia de usuario satisfactoria. Una técnica clave para mejorar la velocidad de carga de una aplicación web es el code splitting o división de bundles. Esta estrategia consiste en dividir el código de tu aplicación en múltiples bundles más pequeños, cargando solo los necesarios para la parte de la aplicación que el usuario está viendo actualmente. Esto reduce significativamente el tiempo de carga inicial y mejora la experiencia general del usuario, especialmente en aplicaciones web grandes y complejas.

I. ¿Qué es Code Splitting?

El code splitting es una técnica de optimización que divide el código de una aplicación en chunks o fragmentos más pequeños. En lugar de cargar todo el código de una vez, solo se cargan los chunks necesarios para la página inicial o la sección que el usuario está viendo. A medida que el usuario interactúa con la aplicación, se cargan los chunks adicionales bajo demanda. Esto resulta en tiempos de carga más rápidos, especialmente en aplicaciones que tienen una gran cantidad de código JavaScript.

I. Beneficios del Code Splitting

- **Rendimiento mejorado:** El tiempo de carga inicial se reduce drásticamente, lo que lleva a una mejor experiencia del usuario.
- **Experiencia de usuario más fluida:** Los usuarios ven contenido más rápido y pueden interactuar con la aplicación antes.
- **Menor consumo de banda ancha:** Se descargan solo los recursos necesarios, ahorrando ancho de banda tanto para el usuario como para el servidor.
- **Mejor SEO:** Las páginas cargan más rápido, lo que mejora el posicionamiento en los motores de búsqueda.
- **Mejor manejo de errores:** Si un chunk falla, el resto de la aplicación puede seguir funcionando.

I. Técnicas de Code Splitting

Import dinámico

El import dinámico es una característica de JavaScript que permite cargar módulos de forma asíncrona. Esto significa que el módulo no se carga hasta que se necesita. Es una

Torna comunitaria: todo organizado que el módulo no se carga hasta que se necesite. Es otra forma sencilla y eficaz de implementar code splitting.

```
const getComponent = () => import('./myComponent');

getComponent().then(module => {
  const MyComponent = module.default;
  ReactDOM.render(, document.getElementById('root'));
});
```

Este ejemplo carga el componente `myComponent` solo cuando se llama a la función `getComponent`.

React.lazy y Suspense

React ofrece `React.lazy` y `Suspense` para implementar code splitting de una forma más declarativa. `React.lazy` permite cargar componentes de forma asíncrona, mientras que `Suspense` proporciona una forma de mostrar un indicador de carga mientras se carga el componente.

```
const MyComponent = React.lazy(() => import('./myComponent'));

function MyPage() {
  return (
    Loading...
  );
}
```

 Compartir en Twitter

 Compartir en Facebook

 Compartir en LinkedIn

 Compartir por WhatsApp



hgaruna

  

¿Te gustó este artículo?

Suscríbete a nuestro boletín para recibir más contenido como este directamente en tu correo.

Tu correo electrónico

Suscribirme

Computer Vision en aplicaciones web

2025-07-28

4 min de lectura

Por hgaruna

La visión por computadora está revolucionando el desarrollo web, permitiendo a las aplicaciones interactuar con el mundo real de una manera nunca antes vista. Esta tecnología, rama de la inteligencia artificial, permite a las computadoras "ver" e interpretar imágenes y videos, abriendo un abanico de posibilidades para mejorar la experiencia del usuario y crear aplicaciones innovadoras. En este artículo, exploraremos cómo la visión por computadora se está integrando en las aplicaciones web y analizaremos algunos ejemplos concretos de su implementación.

I Integración de la Visión por Computadora en Aplicaciones Web

La integración de la visión por computadora en aplicaciones web generalmente se realiza a través de APIs de servicios en la nube, como Google Cloud Vision API, Amazon Rekognition o Microsoft Azure Computer Vision. Estas APIs ofrecen una variedad de funciones, incluyendo detección de objetos, reconocimiento facial, análisis de escenas y extracción de texto de imágenes. El desarrollador puede acceder a estas funciones mediante solicitudes HTTP, enviando la imagen a la API y recibiendo los resultados en formato JSON.

Ventajas de usar APIs en la nube

Utilizar APIs de servicios en la nube presenta varias ventajas significativas:

- **Escalabilidad:** Las APIs se encargan de gestionar la infraestructura necesaria para procesar las imágenes, permitiendo escalar fácilmente la aplicación según la demanda.
- **Precisión:** Los modelos de visión por computadora de estas APIs están entrenados con grandes conjuntos de datos, lo que resulta en una mayor precisión en comparación con modelos entrenados localmente.
- **Facilidad de uso:** Las APIs ofrecen interfaces sencillas y bien documentadas, simplificando la integración en las aplicaciones web.
- **Costo-efectivo:** Generalmente se paga por uso, evitando la necesidad de invertir en hardware y software costosos.

I Ejemplos de Aplicaciones Web con Visión por Computadora

La visión por computadora se aplica en una amplia gama de aplicaciones web. Algunos ejemplos incluyen:

1. **Búsqueda de imágenes inversa:** Permite a los usuarios subir una imagen y encontrar imágenes similares en una base de datos. Esto se logra extrayendo características de la imagen subida y comparándolas con las características de las imágenes en la base de datos.
2. **Análisis de imágenes de productos:** Sitios de comercio electrónico pueden utilizar la visión por

computadora para analizar imágenes de productos y extraer información como el color, la marca o el modelo. Esto facilita la búsqueda y la categorización de productos.

3. **Detección de objetos en tiempo real:** Aplicaciones de realidad aumentada pueden utilizar la visión por computadora para detectar objetos en el entorno del usuario y superponer información adicional en la imagen en tiempo real.
4. **Control de calidad:** En la industria manufacturera, la visión por computadora puede automatizar procesos de control de calidad, detectando defectos en productos o piezas.

Ejemplo de Detección de Objetos con Google Cloud Vision API

El siguiente ejemplo de código JavaScript muestra cómo utilizar la Google Cloud Vision API para detectar objetos en una imagen:

```
// Reemplazar con tu clave de API
const apiKey = 'YOUR_API_KEY';

async function detectObjects(image) {
  const response = await fetch(`https://vision.googleapis.com/v1/images:annotate?
key=${apiKey}`, {
    method: 'POST',
    headers: {
      'Content-Type': 'application/json'
    },
    body: JSON.stringify({
      requests: [
        {
          image: {
            content: image // Imagen en formato base64
          },
          features: [
            {
              type: 'OBJECT_LOCALIZATION'
            }
          ]
        }
      ]
    })
  });
}

const data = await response.json();
return data.responses[0].localizedObjectAnnotations;
}

// Ejemplo de uso
const imageData = 'base64-encoded-image-data'; // Reemplazar con la imagen
detectObjects(imageData)
  .then(objects => {
    console.log(objects); // Mostrar los objetos detectados
  })
  .catch(error => {
    console.error('Error:', error);
  });
}
```

Al utilizar la visión por computadora en aplicaciones web, es crucial tener en cuenta las implicaciones éticas y de privacidad. Es importante:

- Obtener el consentimiento informado del usuario antes de procesar sus imágenes.
- Utilizar los datos de manera responsable y transparente.
- Implementar medidas de seguridad para proteger la privacidad de los usuarios.
- Ser consciente de los posibles sesgos en los modelos de visión por computadora y trabajar para mitigarlos.

La visión por computadora ha transformado la forma en que interactuamos con las tecnologías digitales. Aunque existen desafíos técnicos y éticos, las ventajas que ofrece esta tecnología superan ampliamente los inconvenientes, abriendo un futuro prometedor para la innovación en el desarrollo web.

Conclusión

La visión por computadora está transformando el panorama del desarrollo web, permitiendo la creación de aplicaciones más inteligentes e interactivas. Aunque existen desafíos técnicos y éticos, las ventajas que ofrece esta tecnología superan ampliamente los inconvenientes, abriendo un futuro prometedor para la innovación en el desarrollo web.

#JavaScript

#API

#Seguridad

#IA

Compartir este artículo:

Twitter

Facebook

LinkedIn

WhatsApp



hgaruna

¿Te gustó este artículo?

Suscríbete a nuestro boletín para recibir más contenido como este directamente en tu correo.

Tu correo electrónico

Tu correo electrónico

Suscribirme

Introducción

2025-07-27

5 min de lectura

Por hgaruna

I Introducción

La Content Security Policy (CSP) es un mecanismo de seguridad HTTP que permite a los desarrolladores web controlar los recursos que el navegador puede cargar para una página web dada. Esto ayuda a mitigar los riesgos de ataques como el Cross-Site Scripting (XSS) y otros tipos de inyección de código malicioso. En esencia, CSP define una política de seguridad que el navegador debe aplicar rigurosamente, bloqueando cualquier recurso que no cumpla con las directivas especificadas. Implementar una CSP eficaz es una parte crucial de una estrategia de seguridad web robusta.

I Sección Principal

Una CSP se define mediante un encabezado HTTP, `Content-Security-Policy`, o a través de una etiqueta `meta`. Este encabezado o etiqueta contiene una lista de directivas, cada una especificando qué tipos de recursos se permiten cargar y de dónde. Si un recurso intenta cargarse y no cumple con la política definida, el navegador lo bloqueará, previniendo potenciales ataques. El uso de CSP es altamente recomendado para cualquier sitio web que maneje datos sensibles o información de usuario.

Directivas CSP comunes

Existen varias directivas CSP, cada una con su propio propósito. Algunas de las más comunes incluyen:

- `default-src`: Define una política por defecto para todos los tipos de recursos si no se especifica otra directiva. Es una buena práctica siempre definir esta directiva.
- `script-src`: Especifica las fuentes permitidas para scripts. Esto es crucial para prevenir ataques XSS.
- `style-src`: Especifica las fuentes permitidas para hojas de estilo.
- `img-src`: Especifica las fuentes permitidas para imágenes.
- `font-src`: Especifica las fuentes permitidas para fuentes.
- `object-src`: Especifica las fuentes permitidas para objetos como , y

