Configuración de CDN para Assets Estáticos

Resumen

Esta guía explica cómo configurar un CDN para servir assets estáticos (imágenes, documentos, etc.) de manera optimizada.

Opciones de CDN

Opción 1: AWS S3 + CloudFront (Recomendado)

Ventajas

- Integración nativa con S3
- Caché global con 400+ edge locations
- SSL/TLS automático
- Costo-efectivo

Configuración

1. **Crear Distribución de CloudFront**

```
# Usar AWS CLI o consola web
aws cloudfront create-distribution \
  --origin-domain-name your-bucket.s3.amazonaws.com \
  --default-root-object index.html
```

1. **Configurar Origin Access Identity (OAI)**
   - Restringe acceso directo a S3
   - Solo CloudFront puede acceder al bucket

2. **Configurar Cache Behaviors**

```
{
  "PathPattern": "/public/*",
  "TargetOriginId": "S3-inmova-assets",
  "ViewerProtocolPolicy": "redirect-to-https",
  "CachePolicyId": "658327ea-f89d-4fab-a63d-7e88639e58f6", // CachingOptimized
  "Compress": true
}
```

1. **Configurar Custom Domain**

```
assets.inmova.app
```

Opción 2: Cloudflare R2 + CDN

Ventajas

- Sin cargos por transferencia de datos
- CDN incluido

- Más económico para alto tráfico

Configuración

1. **Crear Bucket en R2**

```
wrangler r2 bucket create inmova-assets
```

1. **Configurar Domain en Cloudflare**
   - Añadir CNAME: `assets.inmova.app`
   - Habilitar proxy (orange cloud)

2. **Configurar Access Keys**

```
R2_ACCOUNT_ID=your_account_id
R2_ACCESS_KEY_ID=your_access_key
R2_SECRET_ACCESS_KEY=your_secret_key
```

Opción 3: Vercel Blob Storage + CDN

Ventajas

- Integración nativa con Vercel
- Configuración cero
- CDN automático

Configuración

```
npm install @vercel/blob
```

```
import { put } from '@vercel/blob';

const blob = await put('filename.jpg', file, {
  access: 'public',
});

console.log(blob.url); // URL con CDN incluido
```

Implementación en Next.js

1. Configuración de next.config.js

```
/** @type {import('next').NextConfig} */
const nextConfig = {
  images: {
    remotePatterns: [
      {
        protocol: 'https',
        hostname: 'your-cloudfront-domain.cloudfront.net',
        pathname: '/**',
      },
      {
        protocol: 'https',
        hostname: 'assets.inmova.app',
        pathname: '/**',
      },
    ],
    // Optimización de imágenes
    formats: ['image/avif', 'image/webp'],
    deviceSizes: [640, 750, 828, 1080, 1200, 1920, 2048, 3840],
    imageSizes: [16, 32, 48, 64, 96, 128, 256, 384],
  },
  // Configurar headers de cache
  async headers() {
    return [
      {
        source: '/public/:path*',
        headers: [
          {
            key: 'Cache-Control',
            value: 'public, max-age=31536000, immutable',
          },
        ],
      },
    ];
  },
};

module.exports = nextConfig;
```

2. Helper para URLs de CDN

```typescript
// lib/cdn-urls.ts

const CDN_BASE_URL = process.env.NEXT_PUBLIC_CDN_URL || '';

/**
 * Genera URL de CDN para un asset
 */
export function getCDNUrl(path: string): string {
  if (!CDN_BASE_URL) {
    return path; // Fallback a URL local
  }

  // Remover slash inicial si existe
  const cleanPath = path.startsWith('/') ? path.slice(1) : path;

  return `${CDN_BASE_URL}/${cleanPath}`;
}

/**
 * Genera URL optimizada de imagen con transformaciones
 */
export function getOptimizedImageUrl(
  path: string,
  options?: {
    width?: number;
    height?: number;
    quality?: number;
    format?: 'webp' | 'avif' | 'jpg' | 'png';
  }
): string {
  const baseUrl = getCDNUrl(path);

  if (!options) return baseUrl;

  // Construir query params para transformaciones
  const params = new URLSearchParams();
  if (options.width) params.set('w', options.width.toString());
  if (options.height) params.set('h', options.height.toString());
  if (options.quality) params.set('q', options.quality.toString());
  if (options.format) params.set('fm', options.format);

  return `${baseUrl}?${params.toString()}`;
}

/**
 * Genera URLs para diferentes tamaños (responsive)
 */
export function getResponsiveImageUrls(path: string): {
  sm: string;
  md: string;
  lg: string;
  xl: string;
} {
  return {
    sm: getOptimizedImageUrl(path, { width: 640, quality: 80, format: 'webp' }),
    md: getOptimizedImageUrl(path, { width: 1024, quality: 80, format: 'webp' }),
    lg: getOptimizedImageUrl(path, { width: 1920, quality: 85, format: 'webp' }),
    xl: getOptimizedImageUrl(path, { width: 3840, quality: 85, format: 'webp' }),
  };
}
```

3. Componente de Imagen Optimizada

```tsx
// components/OptimizedImage.tsx

import Image from 'next/image';
import { getCDNUrl } from '@/lib/cdn-urls';

interface OptimizedImageProps {
  src: string;
  alt: string;
  width?: number;
  height?: number;
  fill?: boolean;
  priority?: boolean;
  className?: string;
}

export function OptimizedImage({
  src,
  alt,
  width,
  height,
  fill = false,
  priority = false,
  className,
}: OptimizedImageProps) {
  const cdnUrl = getCDNUrl(src);

  return (
    <Image
      src={cdnUrl}
      alt={alt}
      width={width}
      height={height}
      fill={fill}
      priority={priority}
      className={className}
      loading={priority ? 'eager' : 'lazy'}
      placeholder="blur"
      blurDataURL="
9awAAAABJRU5ErkJggg=="
    />
  );
}
```

4. Upload con URL de CDN

```typescript
// lib/upload-to-cdn.ts

import { uploadFile } from '@/lib/s3';
import { getCDNUrl } from '@/lib/cdn-urls';

export async function uploadAndGetCDNUrl(
  file: Buffer,
  fileName: string,
  isPublic: boolean = true
): Promise<{ s3Key: string; cdnUrl: string }> {
  // Upload a S3
  const s3Key = await uploadFile(file, fileName, isPublic);

  // Generar URL de CDN
  const cdnUrl = getCDNUrl(`/uploads/${fileName}`);

  return { s3Key, cdnUrl };
}
```

Variables de Entorno

```bash
# CDN Configuration
NEXT_PUBLIC_CDN_URL=https://assets.inmova.app
# o
NEXT_PUBLIC_CDN_URL=https://d1234567.cloudfront.net

# CloudFront (si aplica)
CLOUDFRONT_DISTRIBUTION_ID=E1234567890ABC
CLOUDFRONT_KEY_PAIR_ID=APKA1234567890ABC
CLOUDFRONT_PRIVATE_KEY=-----BEGIN PRIVATE KEY-----\n...

# Cloudflare R2 (si aplica)
R2_ACCOUNT_ID=your_account_id
R2_ACCESS_KEY_ID=your_access_key
R2_SECRET_ACCESS_KEY=your_secret_key
R2_BUCKET_NAME=inmova-assets
R2_PUBLIC_URL=https://assets.inmova.app
```

Optimizaciones Adicionales

1. Compresión de Imágenes

Usar `sharp` para comprimir imágenes antes de subir:

```
import sharp from 'sharp';

export async function optimizeImage(buffer: Buffer): Promise<Buffer> {
  return sharp(buffer)
    .resize(1920, 1080, {
      fit: 'inside',
      withoutEnlargement: true,
    })
    .webp({ quality: 85 })
    .toBuffer();
}
```

2. Lazy Loading

```
<OptimizedImage
  src="/images/hero.jpg"
  alt="Hero"
  fill
  loading="lazy" // Automático para imágenes no prioritarias
/>
```

3. Preload de Assets Críticos

```
// app/layout.tsx
export default function RootLayout({ children }) {
  return (
    <html>
      <head>
        <link
          rel="preload"
          as="image"
          href={getCDNUrl('/images/logo.png')}
        />
      </head>
      <body>{children}</body>
    </html>
  );
}
```

4. Cache Busting

```
export function getCDNUrlWithVersion(path: string): string {
  const version = process.env.NEXT_PUBLIC_BUILD_ID || Date.now();
  const baseUrl = getCDNUrl(path);
  return `${baseUrl}?v=${version}`;
}
```

Monitoreo de Performance

Métricas Clave

1. **Cache Hit Rate**: > 90%
2. **TTFB (Time to First Byte)**: < 200ms
3. **Image Load Time**: < 1s

4. **Data Transfer**: Reducción del 60-70% con compresión

Herramientas

- **CloudWatch** (AWS)
- **Cloudflare Analytics** (Cloudflare)
- **Lighthouse** (Chrome DevTools)
- **WebPageTest**

Costos Estimados

AWS CloudFront

- Primeros 10 TB: $0.085/GB
- HTTPS requests: $0.010/10,000 requests
- **Estimado mensual (100GB, 1M requests)**: ~$10-15/mes

Cloudflare R2

- Storage: $0.015/GB/mes
- Class A Operations: $4.50/millón
- Class B Operations: $0.36/millón
- **Transferencia: GRATIS**
- **Estimado mensual (100GB storage)**: ~$2-5/mes

Vercel Blob

- Primeros 1GB: Gratis
- Adicional: $0.15/GB
- **Estimado mensual (10GB)**: ~$1.50/mes

Conclusión

Recomendación:
- Para aplicaciones en Vercel: **Vercel Blob**
- Para aplicaciones standalone con AWS: **CloudFront + S3**
- Para alto tráfico con presupuesto limitado: **Cloudflare R2**

Referencias

- Next.js Image Optimization (https://nextjs.org/docs/app/building-your-application/optimizing/images)
- AWS CloudFront Documentation (https://docs.aws.amazon.com/cloudfront/)
- Cloudflare R2 Documentation (https://developers.cloudflare.com/r2/)