🚀 完全デプロイ計画：Next.js + Docker + Cloudflare Tunnel

Phase 1: アプリケーションのDocker化と設定ファイルの準備

このフェーズでは、Next.jsを本番環境向けに最適化し、Dockerイメージをビルドできるように準備します。

1.1 Next.js 設定の更新 (`next.config.ts`)

Next.jsの`standalone`出力モードを有効にし、Dockerのビルド成果物を最適化します。

```
// nemuilemon/denno-meishi/denno-meishi-77897f5adefdda842fd8ba0f77b85885ad2887a1/next.config.ts
import type { NextConfig } from "next";

const nextConfig: NextConfig = {
  output: 'standalone', // Docker用の最適化
};

export default nextConfig;
```

1.2 Dockerfileの作成と修正

Prisma Clientの生成ファイル（`src/generated`）のコピーを明示的に追加し、Next.jsアプリの起動に必要な全てのファイルがコンテナに確実に含まれるようにします。

```
# Dockerfile (修正版、抜粋)

# ... (deps, builder ステージは変更なし)

# 実行層 (最小サイズ)
FROM base AS runner
WORKDIR /app

# ... (ユーザー・環境変数設定は変更なし)

# ビルド成果物のコピー
COPY --from=builder /app/public ./public
COPY --from=builder --chown=nextjs:nodejs /app/.next/standalone ./
COPY --from=builder --chown=nextjs:nodejs /app/.next/static ./.next/static
# 【修正点】Prisma Clientの生成物（src/generated）をコピー
COPY --from=builder --chown=nextjs:nodejs /app/src/generated ./src/generated
COPY --from=builder /app/node_modules/.prisma ./node_modules/.prisma
COPY --from=builder /app/prisma ./prisma

USER nextjs
```

```
EXPOSE 3000

ENV PORT=3000
ENV HOSTNAME="0.0.0.0"

CMD ["node", "server.js"]
```

1.3 .dockerignoreの作成

```
# .dockerignore
# ... (既存の内容に加えて以下を確認)
.env*
postgres-data
# ...
```

`.env*`を無視リストに追加し、機密情報がDockerイメージに含まれないようにします。

Phase 2: Docker ComposeとCloudflare Tunnelの統合

Next.jsアプリのポートをホストに公開せず、`cloudflared`コンテナ経由でのみアクセスできるように設定します。

2.1 docker-compose.yml の更新（Next.jsポートの隔離）

`app`サービスから`ports: "3000:3000"`を削除し、`denno_network`内部でのみアクセス可能にします。

```
# docker-compose.yml (修正版、抜粋)
services:
  # ... (db, adminer サービスは変更なし)

  # Next.jsアプリケーション (app)
  app:
    build:
      context: .
      dockerfile: Dockerfile
    container_name: denno_meishi_app
    restart: unless-stopped
    # 【修正点】ポートをホストに公開しないため、portsセクションを削除またはコメントアウト
    # ports:
    #   - "3000:3000"
    environment:
      - NODE_ENV=production
      # DB_PASSWORDを.envから取得
      - DATABASE_URL=postgresql://postgres:${DB_PASSWORD:-
Your_Super_Secret_Password}@db:5432/denno_meishi
      - ADMIN_USERNAME=${ADMIN_USERNAME:-admin}
```

```
      - ADMIN_PASSWORD=${ADMIN_PASSWORD:-admin_password}
    depends_on:
      db:
        condition: service_healthy
    networks:
      - denno_network
    volumes:
      - ./prisma:/app/prisma

  # Cloudflare Tunnel (cloudflared)
  cloudflared:
    image: cloudflare/cloudflared:latest
    container_name: denno_meishi_tunnel
    restart: unless-stopped
    # Cloudflareの設定ファイルと認証情報をボリュームとしてマウント
    command: tunnel --config /etc/cloudflared/config.yml run
    volumes:
      - ./cloudflared-config.yml:/etc/cloudflared/config.yml:ro
      # 【重要】ホストOS上の認証ファイルへの絶対パスを環境変数で指定します
      - ${CLOUDFLARED_CREDENTIALS}:/etc/cloudflared/credentials.json:ro
    networks:
      - denno_network
    depends_on:
      - app

# ... (volumes, networks セクションは変更なし)
```

2.2 cloudflared-config.yml の作成（内部ルーティング）

`cloudflared`がDocker内部ネットワークで`app`サービス名を使ってアクセスするように設定します。

```
# cloudflared-config.yml (修正版)
tunnel: <YOUR_TUNNEL_ID>
credentials-file: /etc/cloudflared/credentials.json # コンテナ内部の
credentials.jsonのパス

ingress:
  - hostname: yourdomain.com
    service: http://app:3000 # 【修正点】サービス名 'app' を使用
  - hostname: www.yourdomain.com
    service: http://app:3000 # 【修正点】サービス名 'app' を使用
  - service: http_status:404
```

2.3 環境変数設定 (`.env`)

`.env`ファイルに、Cloudflare Tunnelの認証情報ファイルへのパスを追加します。

```
# .env (抜粋)

# データベース設定
DB_PASSWORD=your_strong_production_password

# 管理者認証
ADMIN_USERNAME=admin
ADMIN_PASSWORD=your_admin_password

# Next.js設定
DATABASE_URL=postgresql://postgres:your_strong_production_password@db:5432/denno_meishi
NEXT_PUBLIC_SITE_URL=https://yourdomain.com

# 【新規追加】Cloudflare Tunnelの認証ファイルパス
# ※ご自身の環境に合わせて絶対パスを指定してください（例はLinux/VPSの場合）
CLOUDFLARED_CREDENTIALS=/root/.cloudflared/<YOUR_TUNNEL_ID>.json
```

⚙️ Phase 3: Cloudflare Tunnel のセットアップ手順

デプロイ先サーバーで`cloudflared`バイナリを実行し、Cloudflare Dashboardと連携させます。

ステップ 3.1: `cloudflared` のインストールと認証

1. **`cloudflared`のインストール**: デプロイするサーバー（Linux/VPSなど）のOSに合わせて`cloudflared`をインストールします。
2. **ログイン認証**: 以下のコマンドを実行し、ブラウザでCloudflareアカウントにログインし、対象ドメインを選択します。

   ```
   cloudflared tunnel login
   ```

   これにより、**認証ファイル**が`~/.cloudflared/cert.pem`に作成されます。

ステップ 3.2: Tunnelの作成とID取得（Cloudflare Dashboard 操作）

1. **Cloudflare Dashboard**を開き、**Zero Trust** > **Access** > **Tunnels**に移動します。
2. **「Create a Tunnel」**をクリックし、任意の名前（例: `denno-meishi`）を付けてTunnelを作成します。
3. 作成後、画面に表示される**Tunnel ID**を控えます。
4. サーバーに戻り、以下のコマンドを実行してTunnel設定ファイルを作成します。

   ```
   cloudflared tunnel create denno-meishi
   ```

これにより、`~/.cloudflared/<TUNNEL_ID>.json`に**クレデンシャルファイル**が作成されます。このファイルのパスを`.env`の`CLOUDFLARED_CREDENTIALS`に設定します。

ステップ 3.3: DNSレコードの設定（Cloudflare Dashboard 操作）

1. Cloudflare Dashboardで、あなたのドメインの**DNS**管理画面に移動します。
2. **CNAME**レコードを2つ追加します。
   - **タイプ**: `CNAME`
   - **名前**: `yourdomain.com` (`@`) / **ターゲット**: `<TUNNEL_ID>.cfargotunnel.com`
   - **タイプ**: `CNAME`
   - **名前**: `www` / **ターゲット**: `<TUNNEL_ID>.cfargotunnel.com`
   - 💡 **ヒント**: `cloudflared tunnel route dns denno-meishi yourdomain.com`を実行することで、DNS設定を自動化することもできます。

☑ Phase 4: ローカルでの最終テストとデプロイ

ステップ 4.1: 初回セットアップコマンドの実行

1. **環境変数の準備**: `.env.example`をコピーし、**Phase 2.3**に従って`.env`を編集します。

```
cp .env.example .env
# .envファイルとcloudflared-config.ymlを編集
```

2. **Dockerイメージのビルド**:

```
docker-compose build
```

3. **コンテナの起動**:

```
docker-compose up -d
```

4. **データベースマイグレーション**: DBが`service_healthy`になるまで数秒待ってから実行します。

```
docker-compose exec app npx prisma migrate deploy
```

ステップ 4.2: 動作確認

1. **ログの確認**: `app`サービスと`cloudflared`サービスがエラーなく起動していることを確認します。

```
docker-compose logs -f
```

2. **サイトアクセス**: 設定したURL（`https://yourdomain.com`）にアクセスし、ポートフォリオサイトが表示され、コンタクトフォームの送信（APIエンドポイント）が正常に機能することを確認します。

ステップ 4.3: 停止コマンド

```
# すべてのサービスを停止し、コンテナを削除
docker-compose down

# ボリューム（データベースデータ）も含めて完全に削除する場合
docker-compose down -v
```