cafeMuji 注文管理システム デプロイ手順書

1. デプロイ概要

1.1 デプロイ環境

- **ホスティングサービス**: Render.com
- **アプリケーションタイプ**: Web Service
- **ランタイム**: Python 3.x
- **データベース**: SQLite（本番ではPostgreSQL推奨）

1.2 デプロイの流れ

```
[ローカル開発] → [Gitリポジトリ] → [Render.com] → [本番環境]
      ↓              ↓              ↓              ↓
   開発・テスト    コード管理      自動デプロイ    本番運用
```

2. 事前準備

2.1 必要なアカウント

- **GitHub**: ソースコード管理
- **Render.com**: ホスティングサービス
- **Django**: フレームワーク

2.2 ローカル環境の準備

```
# Python環境の確認
python --version  # Python 3.x以上

# 仮想環境の作成
python -m venv venv

# 仮想環境の有効化
source venv/bin/activate  # macOS/Linux
# venv\Scripts\activate  # Windows

# 依存関係のインストール
pip install -r requirements.txt

# Djangoの動作確認
python manage.py runserver
```

2.3 Gitリポジトリの準備

```
# Gitリポジトリの初期化
git init

# リモートリポジトリの追加
git remote add origin https://github.com/username/cafeMuji.git
```

```
# 初回コミット
git add .
git commit -m "Initial commit: cafeMuji注文管理システム"

# リモートへのプッシュ
git push -u origin main
```

3. Render.comでのデプロイ

3.1 Render.comアカウント作成

1. Render.comにアクセス
2. GitHubアカウントでサインアップ
3. メールアドレスの確認

3.2 新しいWeb Serviceの作成

1. **Dashboard** → **New +** → **Web Service**
2. **Connect a repository**でGitHubリポジトリを選択
3. **cafeMuji**リポジトリを選択

3.3 サービス設定

```
Name: cafeMuji-order-system
Environment: Python 3
Region: Frankfurt (EU Central)
Branch: main
Root Directory: ./
Build Command: pip install -r requirements.txt
Start Command: gunicorn config.wsgi:application
```

3.4 環境変数の設定

```
Key: PYTHON_VERSION
Value: 3.11.0

Key: DJANGO_SETTINGS_MODULE
Value: config.settings

Key: SECRET_KEY
Value: [生成された秘密鍵]

Key: DEBUG
Value: False

Key: ALLOWED_HOSTS
Value: .onrender.com
```

3.5 自動デプロイの設定

- **Auto-Deploy**: Yes
- **Branch**: main
- **Deploy on Push**: Yes

4. 設定ファイルの準備

4.1 Procfile

```
web: gunicorn config.wsgi:application
```

4.2 requirements.txt

```
asgiref==3.8.1
Django==5.2.1
gunicorn==23.0.0
packaging==25.0
sqlparse==0.5.3
```

4.3 Django設定ファイルの調整

```
# config/settings.py

# 本番環境用の設定
DEBUG = False

ALLOWED_HOSTS = [
    '.onrender.com',
    'localhost',
    '127.0.0.1',
]

# 静的ファイルの設定
STATIC_URL = '/static/'
STATIC_ROOT = os.path.join(BASE_DIR, 'staticfiles')

# セキュリティ設定
SECURE_SSL_REDIRECT = True
SESSION_COOKIE_SECURE = True
CSRF_COOKIE_SECURE = True

# ログ設定
LOGGING = {
    'version': 1,
    'disable_existing_loggers': False,
    'handlers': {
        'console': {
            'class': 'logging.StreamHandler',
        },
    },
    'root': {
        'handlers': ['console'],
        'level': 'INFO',
    },
}
```

5. デプロイ実行

5.1 初回デプロイ

1. **Create Web Service**をクリック
2. ビルドプロセスの開始
3. ビルド完了まで待機（5-10分）

5.2 ビルドログの確認

```
Building application...
Installing dependencies...
Installing asgiref==3.8.1
Installing Django==5.2.1
Installing gunicorn==23.0.0
Installing packaging==25.0
Installing sqlparse==0.5.3
Building static files...
Starting application...
```

5.3 デプロイ完了の確認

- **Status**: Live
- **URL**: https://cafemuji-order-system.onrender.com
- **Last Deploy**: [日時]

6. データベースの設定

6.1 SQLite（開発環境）

```
# config/settings.py
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.sqlite3',
        'NAME': BASE_DIR / 'db.sqlite3',
    }
}
```

6.2 PostgreSQL（本番環境推奨）

```
# config/settings.py
import os

DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.postgresql',
        'NAME': os.environ.get('POSTGRES_DB'),
        'USER': os.environ.get('POSTGRES_USER'),
        'PASSWORD': os.environ.get('POSTGRES_PASSWORD'),
        'HOST': os.environ.get('POSTGRES_HOST'),
        'PORT': os.environ.get('POSTGRES_PORT', '5432'),
    }
}
```

6.3 データベースの初期化

```
# マイグレーションの実行
python manage.py migrate

# スーパーユーザーの作成
python manage.py createsuperuser

# 初期データの投入（必要に応じて）
python manage.py loaddata initial_data.json
```

7. 静的ファイルの設定

7.1 静的ファイルの収集

```
# 静的ファイルの収集
python manage.py collectstatic --noinput

# 静的ファイルの確認
ls staticfiles/
```

7.2 静的ファイルの設定

```
# config/settings.py
STATIC_URL = '/static/'
STATIC_ROOT = os.path.join(BASE_DIR, 'staticfiles')

STATICFILES_DIRS = [
    os.path.join(BASE_DIR, 'static'),
]
```

7.3 静的ファイルの配信

```
# config/urls.py
from django.conf import settings
from django.conf.urls.static import static

urlpatterns = [
    # 既存のURL設定
]

if settings.DEBUG:
    urlpatterns += static(settings.STATIC_URL, document_root=settings.STATIC_ROOT)
```

8. セキュリティ設定

8.1 環境変数による秘密鍵管理

```
# config/settings.py
import os

SECRET_KEY = os.environ.get('SECRET_KEY', 'django-insecure-default-key')
```

8.2 HTTPS設定

```
# config/settings.py
SECURE_SSL_REDIRECT = True
SESSION_COOKIE_SECURE = True
CSRF_COOKIE_SECURE = True
SECURE_BROWSER_XSS_FILTER = True
SECURE_CONTENT_TYPE_NOSNIFF = True
```

8.3 セキュリティヘッダー

```
# config/settings.py
SECURE_HSTS_SECONDS = 31536000
SECURE_HSTS_INCLUDE_SUBDOMAINS = True
SECURE_HSTS_PRELOAD = True
```

9. 監視・ログ

9.1 ログ設定

```
# config/settings.py
LOGGING = {
    'version': 1,
    'disable_existing_loggers': False,
    'formatters': {
        'verbose': {
            'format': '{levelname} {asctime} {module} {process:d} {thread:d} {message}',
            'style': '{',
        },
    },
    'handlers': {
        'console': {
            'class': 'logging.StreamHandler',
            'formatter': 'verbose',
        },
    },
    'root': {
        'handlers': ['console'],
        'level': 'INFO',
    },
    'loggers': {
        'django': {
            'handlers': ['console'],
            'level': 'INFO',
            'propagate': False,
        },
    },
}
```

9.2 ヘルスチェック

```
# config/urls.py
from django.http import HttpResponse

def health_check(request):
    return HttpResponse("OK")

urlpatterns = [
    path('health/', health_check, name='health_check'),
    # 既存のURL設定
]
```

10. 運用・メンテナンス

10.1 定期メンテナンス

```
# 依存関係の更新
pip install --upgrade -r requirements.txt

# データベースのバックアップ
python manage.py dumpdata > backup_$(date +%Y%m%d).json

# ログファイルの確認
tail -f logs/django.log
```

10.2 パフォーマンス監視

- **Render Dashboard**: CPU、メモリ使用量の確認
- **応答時間**: 各エンドポイントの応答時間監視
- **エラー率**: エラーログの確認

10.3 スケーリング

- **Auto-Scaling**: 必要に応じて有効化
- **インスタンス数**: 負荷に応じて調整
- **リソース割り当て**: CPU・メモリの調整

11. トラブルシューティング

11.1 よくある問題と解決方法

ビルドエラー

```
# 依存関係の問題
pip install --upgrade pip
pip install -r requirements.txt

# Pythonバージョンの問題
python --version
# requirements.txtでバージョンを指定
```

起動エラー

```
# ポートの競合
lsof -i :8000
kill -9 [PID]

# 環境変数の問題
echo $DJANGO_SETTINGS_MODULE
```

データベースエラー

```
# マイグレーションの問題
python manage.py makemigrations
python manage.py migrate

# データベースの確認
python manage.py dbshell
```

11.2 ログの確認方法

```
# Render Dashboardでのログ確認
# Logsタブ → リアルタイムログ

# ローカルでのログ確認
tail -f logs/django.log
```

12. バックアップ・復旧

12.1 データベースのバックアップ

```
# 全データのバックアップ
python manage.py dumpdata > full_backup.json

# アプリ別バックアップ
```

```
python manage.py dumpdata food > food_backup.json
python manage.py dumpdata ice > ice_backup.json
python manage.py dumpdata shavedice > shavedice_backup.json
```

12.2 復旧手順

```
# データベースの復旧
python manage.py loaddata full_backup.json

# 特定アプリの復旧
python manage.py loaddata food_backup.json
```

12.3 自動バックアップ

```
# cronジョブでの自動バックアップ
0 2 * * * cd /path/to/cafeMuji && python manage.py dumpdata > backup_$(date +\%Y\%m\%d).json
```

13. 更新・デプロイ

13.1 コード更新の流れ

```
# ローカルでの変更
git add .
git commit -m "機能追加: 新しいメニュー対応"
git push origin main

# Render.comでの自動デプロイ
# プッシュ後、自動的にデプロイが開始される
```

13.2 手動デプロイ

1. Render Dashboard → **Manual Deploy**
2. **Deploy latest commit**をクリック
3. デプロイ完了まで待機

13.3 ロールバック

1. Render Dashboard → **Deploys**
2. 前回のデプロイを選択
3. **Rollback**をクリック

14. パフォーマンス最適化

14.1 データベース最適化

```
# インデックスの追加
class FoodOrder(models.Model):
    class Meta:
        indexes = [
            models.Index(fields=['status', 'is_completed']),
            models.Index(fields=['clip_color', 'clip_number']),
        ]
```

14.2 キャッシュ設定

```
# config/settings.py
CACHES = {
    'default': {
        'BACKEND': 'django.core.cache.backends.locmem.LocMemCache',
        'LOCATION': 'unique-snowflake',
    }
}
```

14.3 静的ファイル最適化

```
# CSS/JSの最小化
pip install django-compressor

# 画像の最適化
pip install Pillow
```

作成日: 2025年8月 **作成者**: 村岡 優次郎 **バージョン**: 1.0