CI/CD Guide for Sports Analytics Project

What is CI/CD?

CI/CD stands for Continuous Integration and Continuous Deployment. It's a set of practices that helps you automate the process of building, testing, and deploying your code.

- **Continuous Integration (CI)**: Automatically building and testing code whenever changes are pushed
- **Continuous Deployment (CD)**: Automatically deploying code to production when it passes tests

Current Setup Analysis

Your project already has some basic CD through:

- Netlify: Auto-deploys frontend when you push to main
- Render: Auto-deploys backend when you push to main

Free CI/CD Services for Your Stack

1. GitHub Actions (CI)

Perfect for running tests and checks before deployment.

Example workflow for your backend:

```yaml
# .github/workflows/backend-tests.yml
name: Backend Tests

on:
  push:
    branches: [ main ]
    paths:
      - 'backend/**'
  pull_request:
    branches: [ main ]
    paths:
      - 'backend/**'

jobs:
  test:
    runs-on: ubuntu-latest

    steps:
    - uses: actions/checkout@v2

    - name: Set up Python
      uses: actions/setup-python@v2
      with:
        python-version: '3.10'

    - name: Install dependencies
      run: |
        cd backend
        pip install -r requirements.txt
        pip install pytest

    - name: Run tests
      env:
        DATABASE_URL: ${{ secrets.TEST_DATABASE_URL }}
      run: |
        cd backend
        pytest
```

Example workflow for your frontend:

```
# .github/workflows/frontend-tests.yml
name: Frontend Tests

on:
  push:
    branches: [ main ]
    paths:
      - 'src/**'
  pull_request:
    branches: [ main ]
    paths:
      - 'src/**'

jobs:
  test:
    runs-on: ubuntu-latest

    steps:
    - uses: actions/checkout@v2

    - name: Setup Node.js
      uses: actions/setup-node@v2
      with:
        node-version: '18'

    - name: Install dependencies
      run: npm install

    - name: Run tests
      run: npm test
```

2. Netlify (CD for Frontend)

You're already using Netlify's CD features. Enhance them with:

1. Branch Deployments:

```
# netlify.toml
[build]
  command = "npm run build"
  publish = "dist"

[context.production]
  environment = { NODE_VERSION = "18.0.0" }

[context.deploy-preview]
  command = "npm run build:preview"
```

2. Deploy Previews:

- Automatically creates preview URLs for pull requests
- Enables visual testing before merging

3. Render (CD for Backend)

Enhance your existing Render setup with:

1. Branch Environments:

- Create separate services for staging/development
- Use different environment variables per environment

2. Health Checks:

```
# backend/main.py
@app.get("/health")
async def health_check():
    try:
        # Test database connection
        db = SessionLocal()
        db.execute("SELECT 1")
        return {"status": "healthy"}
    except Exception as e:
        return {"status": "unhealthy", "error": str(e)}
```

Recommended CI/CD Pipeline

1. **Development Flow**:

```
Local Development → Git Push → GitHub Actions Tests → Deploy Preview → Production Deploy
```

2. **Branch Strategy**:

```
main          (production)
├── staging   (pre-production tests)
└── dev       (development branch)
    └── feature branches
```

3. **Quality Gates**:

- Linting (pre-commit)
- Unit tests (GitHub Actions)
- Integration tests (GitHub Actions)
- Preview deployments (Netlify)
- Health checks (Render)

Implementation Steps

1. **Set Up Testing**:

```
# Backend tests
cd backend
pip install pytest
# Create tests/test_main.py

# Frontend tests
npm install --save-dev vitest
# Add test scripts to package.json
```

2. **Add GitHub Actions**:
    - Create `.github/workflows/` directory
    - Add workflow files for frontend and backend
3. **Configure Branch Protection**:
    - Go to GitHub repository settings
    - Require passing checks before merging
    - Enable required reviews
4. **Environment Setup**:

```
# Create environment files
backend/.env.test
backend/.env.staging
backend/.env.production

.env.test
.env.staging
.env.production
```

Best Practices

1. **Security**:
    - Store secrets in GitHub Secrets
    - Use environment variables for configuration
    - Never commit sensitive data
2. **Testing**:
    - Write tests for new features
    - Include both unit and integration tests
    - Test database migrations
3. **Monitoring**:
    - Use health check endpoints
    - Set up error tracking
    - Monitor deployment success rates

Resources

Free Tools for Your Stack:

- GitHub Actions: CI pipelines
- Netlify: Frontend CD
- Render: Backend CD
- GitHub: Branch protection and code reviews
- Pytest: Backend testing
- Vitest: Frontend testing