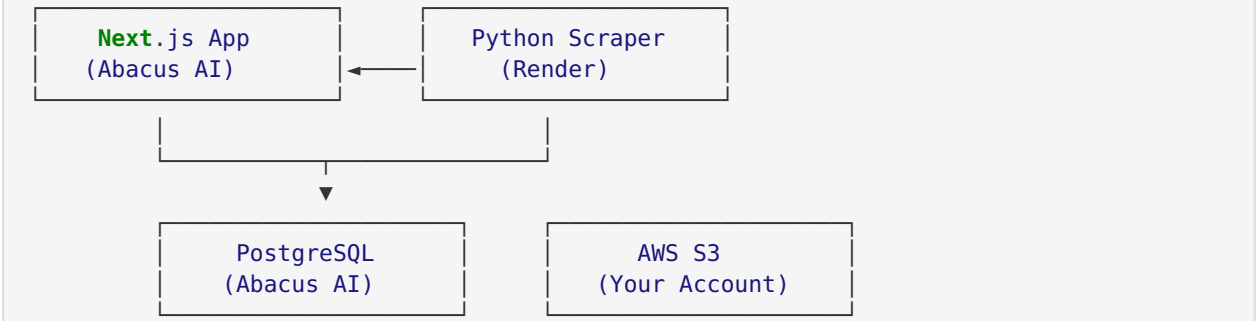
Python Scraper Service - Deployment Guide

📋 Overview

The Python Scraper Service is a Flask-based API that:
- Scrapes NBA and historical basketball player data
- Downloads and uploads player images to S3
- Populates the PostgreSQL database with elite shooter profiles
- Provides API endpoints for triggering scrapes and managing data
- Replaces the old `python-backend` service on Render

Key Features:
- NBA.com Stats API integration
- Basketball-Reference.com web scraping
- Image processing and S3 storage
- Automated database backups
- RESTful API with authentication

🏗️ Architecture



📁 Directory Structure

```
python-scraper/
├── app.py                          # Flask API server
├── main.py                         # Main scraping pipeline
├── config.py                       # Configuration
├── database.py                     # Database operations
├── database_images.py              # Image database operations
├── requirements.txt                # Python dependencies (19 packages)
├── Procfile                        # Render start command
├── runtime.txt                     # Python version (3.11.6)
├── .env                            # Environment variables (DO NOT COMMIT)
├── .env.example                    # Environment template
├── .gitignore                      # Git ignore rules
│
├── scrapers/                       # Web scrapers
│   ├── nba_scraper.py              # NBA.com Stats API
│   ├── basketball_reference_scraper.py  # Historical data
│   ├── image_scraper.py            # Image downloader
│   └── video_frame_extractor.py    # Video processing
│
├── storage/                        # S3 integration
│   └── s3_uploader.py              # AWS S3 uploader
│
├── backup/                         # Database backup system
│   ├── backup_manager.py           # Backup orchestration
│   ├── backup_config.py            # Backup configuration
│   └── scheduler.py                # Automated scheduling
│
├── utils/                          # Utilities
│   └── data_cleaner.py             # Data validation
│
└── migrations/                     # Database migrations
    ├── apply_indexes.py            # Index management
    └── create_indexes.sql          # SQL indexes
```

🔑 Prerequisites

1. AWS Account Setup

You **MUST** have your own AWS account with S3 configured:

Steps:

1. **Create AWS Account** (if you don't have one)
   - Go to https://aws.amazon.com/
   - Sign up for free tier

2. **Create IAM User**
   - Go to AWS Console > IAM > Users
   - Click "Create User"
   - User name: `basketball-scraper`
   - Enable "Programmatic access"

3. **Attach S3 Permissions**
   - Attach policy: `AmazonS3FullAccess`
   - Or create custom policy with S3 read/write permissions

4. **Get Access Keys**
   - After user creation, go to “Security credentials”
   - Click “Create access key”
   - Choose “Application running on AWS compute service”
   - **SAVE THESE CREDENTIALS** (you’ll need them later):
     - `AWS_ACCESS_KEY_ID`
     - `AWS_SECRET_ACCESS_KEY`
5. **Create S3 Bucket**
   - Go to S3 > Create Bucket
   - Bucket name: `basketball-shooters-media` (or your preferred name)
   - Region: `us-west-2` (or your preferred region)
   - Block all public access: **OFF** (images need to be publicly readable)
   - Enable versioning: **Optional**
   - Click “Create bucket”
6. **Configure Bucket Policy** (for public read access)
   - Go to your bucket > Permissions > Bucket Policy
   - Add this policy (replace `YOUR-BUCKET-NAME`):

```
json
  {
    "Version": "2012-10-17",
    "Statement": [
      {
        "Sid": "PublicReadGetObject",
        "Effect": "Allow",
        "Principal": "*",
        "Action": "s3:GetObject",
        "Resource": "arn:aws:s3:::YOUR-BUCKET-NAME/*"
      }
    ]
  }
```

2. Render Account

- Sign up at https://render.com (Free tier available)
- Connect your GitHub account

3. Database URL

- You already have this from the frontend deployment
- It’s in `/home/ubuntu/basketball_app/basketball-analysis/.env`

🚀 Deployment Steps

Step 1: Prepare the Code

The code is already prepared in `/home/ubuntu/basketball_app/python-scraper/`

Key files created:
- ✅ `.env` - Environment variables (configured)
- ✅ `runtime.txt` - Python 3.11.6

- ✅ `.gitignore` - Git ignore rules
- ✅ `Procfile` - Render start command

Step 2: Update Environment Variables

Edit `/home/ubuntu/basketball_app/python-scraper/.env`:

```
# 1. Add your AWS credentials
AWS_ACCESS_KEY_ID=YOUR_AWS_ACCESS_KEY_HERE
AWS_SECRET_ACCESS_KEY=YOUR_AWS_SECRET_KEY_HERE
AWS_REGION=us-west-2
S3_BUCKET_NAME=basketball-shooters-media

# 2. Update frontend URL (after frontend is deployed)
NEXTJS_API_URL=https://your-app.abacus.ai/api

# 3. Database URL is already configured (from frontend)
# DATABASE_URL="postgresql://..." (already set)

# 4. API key is already generated
# API_SECRET_KEY=MSR9VABa1ETXlBkLoXzGpLfnGXsNDQm5C7VcrM-GnjI
```

Step 3: Commit and Push to GitHub

```
cd /home/ubuntu/basketball_app

# Add python-scraper files
git add python-scraper/

# Commit changes
git commit -m "Add Python scraper service for deployment"

# Push to GitHub
git push origin main
```

⚠️ **IMPORTANT:** Make sure `.env` is in `.gitignore` and NOT committed!

Step 4: Deploy to Render (Replacing Old Backend)

Option A: Update Existing Service

1. **Go to Render Dashboard**
   - Navigate to https://dashboard.render.com
   - Find your existing service: `basketball-analysis-backend`

2. **Update Settings**
   - Go to Settings
   - Update “Root Directory”: `python-scraper`
   - Update “Build Command”: `pip install -r requirements.txt`
   - Update “Start Command”: `gunicorn app:app --bind 0.0.0.0:$PORT --workers 2 --timeout 120`

3. **Update Environment Variables**
   - Go to Environment tab
   - Add/Update these variables:

```
DATABASE_URL=postgresql://
```

role_98aaf8ef8:A0YpOM7klQCJsj6RHvTtg2wgXkzmmGoT@db-98aaf8ef8.db003.hosteddb.reai.io:5432/98aaf8ef8?connect_timeout=15

API_SECRET_KEY=MSR9VABa1ETXlBkLoXzGpLfnGXsNDQm5C7VcrM-GnjI

AWS_ACCESS_KEY_ID=[Your AWS Access Key]
AWS_SECRET_ACCESS_KEY=[Your AWS Secret Key]
AWS_REGION=us-west-2
S3_BUCKET_NAME=basketball-shooters-media

NEXTJS_API_URL=https://your-app.abacus.ai/api

DEBUG=false
FLASK_ENV=production

BACKUP_BUCKET=basketball-shooters-media
LOCAL_BACKUP_DIR=/tmp/db_backups
BACKUP_ENCRYPTION_ENABLED=false
```

1. **Trigger Deploy**
   - Click “Manual Deploy” > “Deploy latest commit”
   - Wait for deployment to complete (5-10 minutes)

### Option B: Create New Service

1. **Create New Web Service**
   - Go to Render Dashboard
   - Click “New +” > “Web Service”
   - Connect your GitHub repository

2. **Configure Service**
   - Name: `basketball-scraper-service`
   - Region: `Oregon (US West)` or closest to you
   - Branch: `main`
   - Root Directory: `python-scraper`
   - Runtime: `Python 3`
   - Build Command: `pip install -r requirements.txt`
   - Start Command: `gunicorn app:app --bind 0.0.0.0:$PORT --workers 2 --timeout 120`

3. **Add Environment Variables**
   - Same as Option A above

4. **Create Web Service**
   - Click “Create Web Service”
   - Wait for deployment (5-10 minutes)

5. **Delete Old Service** (after testing)
   - Go to old `basketball-analysis-backend` service
   - Settings > Delete Service

## Step 5: Verify Deployment

Once deployed, test the following endpoints:
```

```
# Replace with your actual Render URL
SCRAPER_URL="https://basketball-scraper-service.onrender.com"
API_KEY="MSR9VABa1ETXlBkLoXzGpLfnGXsNDQm5C7VcrM-GnjI"

# 1. Health check (public endpoint)
curl $SCRAPER_URL/

# Expected: {"service":"Basketball Shooter
Scraper","status":"running","timestamp":"..."}

# 2. Detailed health (public endpoint)
curl $SCRAPER_URL/health

# Expected: {"status":"healthy","database":"connected","timestamp":"..."}

# 3. List shooters (public endpoint)
curl "$SCRAPER_URL/api/shooters?limit=5"

# Expected: {"success":true,"count":0,"shooters":[]} (empty at first)
```

🌱 Seeding the Database

After deployment, run the scraper to populate the database:

Method 1: Via API (Recommended)

```
SCRAPER_URL="https://your-scraper.onrender.com"
API_KEY="MSR9VABa1ETXlBkLoXzGpLfnGXsNDQm5C7VcrM-GnjI"

# 1. Scrape NBA players (100 players)
curl -X POST "$SCRAPER_URL/api/scrape/nba" \
  -H "X-API-Key: $API_KEY" \
  -H "Content-Type: application/json" \
  -d '{"limit": 100}'

# Expected: {"success":true,"message":"Scraped 100 NBA players","count":100}

# 2. Scrape historical players (50 players)
curl -X POST "$SCRAPER_URL/api/scrape/historical" \
  -H "X-API-Key: $API_KEY" \
  -H "Content-Type: application/json" \
  -d '{"limit": 50}'

# Expected: {"success":true,"message":"Scraped 50 historical players","count":50}

# 3. Download images for shooters
curl -X POST "$SCRAPER_URL/api/scrape/images" \
  -H "X-API-Key: $API_KEY" \
  -H "Content-Type: application/json" \
  -d '{"limit": 50}'

# Expected: {"success":true,"message":"Image pipeline completed"}

# 4. Verify data
curl "$SCRAPER_URL/api/shooters?limit=10"

# Expected: {"success":true,"count":10,"shooters":[...]}
```

Method 2: Render Shell (Alternative)

If you need to run the scraper directly:

1. Go to Render Dashboard > Your Service
2. Click “Shell” tab
3. Run commands:

```
# Scrape NBA players
python main.py nba 100

# Scrape historical players
python main.py historical 50

# Run full pipeline
python main.py full

# Download images
python main.py images 50
```

🔧 API Endpoints Reference

Public Endpoints (No Auth Required)

Endpoint	Method	Description
`/`	GET	Basic health check
`/health`	GET	Detailed health status
`/api/shooters`	GET	List all shooters (paginated)
`/api/shooters/<name>`	GET	Get shooter by name

Protected Endpoints (API Key Required)

Include API key in headers: `X-API-Key: your-api-key`

Endpoint	Method	Description
`/api/scrape/nba`	POST	Scrape NBA players
`/api/scrape/historical`	POST	Scrape historical players
`/api/scrape/full`	POST	Run full scraping pipeline
`/api/scrape/images`	POST	Download and upload images
`/api/backup/daily`	POST	Create daily backup
`/api/backup/full`	POST	Full database backup
`/api/backup/list`	GET	List available backups
`/api/backup/restore`	POST	Restore from backup
`/webhook/nextjs`	POST	Webhook for Next.js triggers

Example API Calls

```
# List shooters with pagination
curl "$SCRAPER_URL/api/shooters?limit=20&offset=0"

# Get specific shooter
curl "$SCRAPER_URL/api/shooters/Stephen%20Curry"

# Trigger NBA scrape
curl -X POST "$SCRAPER_URL/api/scrape/nba" \
  -H "X-API-Key: $API_KEY" \
  -H "Content-Type: application/json" \
  -d '{"limit": 50}'

# Create backup
curl -X POST "$SCRAPER_URL/api/backup/full" \
  -H "X-API-Key: $API_KEY"
```

🔐 Security Considerations

API Key Management

1. **Keep API Key Secret**
   - Never commit `.env` to Git
   - Store in Render environment variables
   - Rotate periodically

2. **Share with Frontend**
   - Add API key to frontend environment variables
   - Use for triggering scrapes from admin panel

3. **Regenerate if Compromised**
   ```bash
   ```

```
    # Generate new key
    python3 -c "import secrets; print(secrets.token_urlsafe(32))"

# Update in Render environment
# Update in frontend .env
```

AWS Security

1. **IAM Best Practices**
   - Use least-privilege access
   - Don't use root account credentials
   - Enable MFA on AWS account

2. **S3 Bucket Security**
   - Public read only (not write)
   - Enable versioning for backups
   - Set lifecycle policies for cost savings

3. **Credential Rotation**
   - Rotate AWS access keys every 90 days
   - Use AWS Secrets Manager for production

📊 Monitoring and Logs

View Logs in Render

1. Go to Render Dashboard > Your Service
2. Click "Logs" tab
3. Monitor real-time logs
4. Filter by log level (INFO, WARNING, ERROR)

Key Log Patterns

```
# Successful scrape
INFO     | scrapers.nba_scraper:scrape_nba_players:45 - Scraped 100 NBA players

# Database connection
INFO     | database:test_connection:332 - Database connection successful

# Image upload
INFO     | storage.s3_uploader:upload_to_s3:67 - Uploaded image to S3: s3://bucket/pat
h

# Error patterns
ERROR    | main:run_full_scrape:89 - NBA scraping failed: Connection timeout
```

Set Up Alerts

1. **Render Notifications**
   - Go to Service > Settings
   - Enable "Deploy Notifications"
   - Add webhook URL or email

2. **Health Check Monitoring**
   - Use UptimeRobot or similar
   - Monitor `https://your-scraper.onrender.com/health`
   - Alert if status != “healthy”

🔄 Scheduled Scraping

Option 1: Render Cron Jobs

1. Create a new Cron Job in Render:
   - Name: `daily-nba-scrape`
   - Schedule: `0 6 * * *` (6 AM daily)
   - Command: `python main.py nba 100`

2. Create another for historical:
   - Name: `weekly-historical-scrape`
   - Schedule: `0 6 * * 0` (6 AM Sundays)
   - Command: `python main.py historical 50`

Option 2: External Cron Service

Use a service like cron-job.org or EasyCron:

```
# Daily NBA scrape (6 AM)
curl -X POST "https://your-scraper.onrender.com/api/scrape/nba" \
  -H "X-API-Key: your-key" \
  -d '{"limit": 100}'

# Weekly historical scrape (Sundays 6 AM)
curl -X POST "https://your-scraper.onrender.com/api/scrape/historical" \
  -H "X-API-Key: your-key" \
  -d '{"limit": 50}'
```

Option 3: Frontend Trigger

Add an admin panel in your Next.js app to trigger scrapes manually or scheduled.

🐛 Troubleshooting

Database Connection Failed

Symptom: `{"status":"degraded","database":"disconnected"}`

Solutions:
1. Check DATABASE_URL in Render environment variables
2. Verify Abacus AI database is accessible from Render
3. Check network timeout (increase `connect_timeout` in DATABASE_URL)
4. Verify PostgreSQL credentials are correct

S3 Upload Failed

Symptom: `ERROR | storage.s3_uploader:upload_to_s3:67 - S3 upload failed: Access Denied`

Solutions:
1. Verify AWS credentials are correct
2. Check IAM user has S3 write permissions

3. Verify bucket name is correct
4. Check bucket region matches AWS_REGION

Scraper Timeout

Symptom: `ERROR | main:run_full_scrape:89 - NBA scraping failed: Connection timeout`

Solutions:
1. Increase timeout in Procfile: `--timeout 180`
2. Reduce scrape limit (try 50 instead of 100)
3. Add retry logic in scraper code
4. Check if source website is accessible

Module Not Found

Symptom: `ModuleNotFoundError: No module named 'X'`

Solutions:
1. Verify requirements.txt includes all dependencies
2. Check Python version matches runtime.txt (3.11.6)
3. Clear build cache in Render and redeploy
4. Check for typos in import statements

Out of Memory

Symptom: Service crashes during image processing

Solutions:
1. Upgrade Render plan (more RAM)
2. Reduce batch size for image processing
3. Use `opencv-python-headless` (already in requirements.txt)
4. Process images in smaller batches

Render Free Tier Limitations

Issues:
- Service spins down after 15 minutes of inactivity
- First request after spin-down is slow (30-60 seconds)

Solutions:
1. Upgrade to paid plan ($7/month) for always-on
2. Use UptimeRobot to ping service every 14 minutes
3. Accept cold start delays for free tier

📝 Environment Variables Checklist

Before deploying, ensure these are set in Render:

```
✅ DATABASE_URL              # PostgreSQL connection
✅ API_SECRET_KEY            # API authentication
✅ AWS_ACCESS_KEY_ID         # AWS credentials
✅ AWS_SECRET_ACCESS_KEY     # AWS credentials
✅ AWS_REGION                # AWS region (us-west-2)
✅ S3_BUCKET_NAME            # S3 bucket name
✅ NEXTJS_API_URL            # Frontend URL
✅ DEBUG                     # false
✅ FLASK_ENV                 # production
✅ BACKUP_BUCKET             # Backup bucket (same as S3_BUCKET_NAME)
✅ LOCAL_BACKUP_DIR          # /tmp/db_backups
✅ BACKUP_ENCRYPTION_ENABLED # false
```

🎯 Post-Deployment Checklist

- [ ] Service deployed successfully on Render
- [ ] Health check endpoint returns `{"status":"healthy"}`
- [ ] Database connection working
- [ ] AWS S3 credentials configured
- [ ] API key authentication working
- [ ] Run initial NBA scrape (100 players)
- [ ] Run initial historical scrape (50 players)
- [ ] Run image download pipeline
- [ ] Verify shooters in database
- [ ] Verify images uploaded to S3
- [ ] Test API endpoints from frontend
- [ ] Set up monitoring/alerts
- [ ] Configure scheduled scraping (optional)
- [ ] Update frontend with scraper URL
- [ ] Delete old python-backend service (if using new service)

🔗 Related Documentation

- Frontend Deployment Guide (/home/ubuntu/basketball_app/DEPLOYMENT_ANALYSIS.md)
- GitHub Updates Analysis (/home/ubuntu/basketball_app/GITHUB_UPDATES_ANALYSIS.md)
- Python Scraper README (/home/ubuntu/basketball_app/python-scraper/README.md)

📞 Support

If you encounter issues:

1. Check Render logs first
2. Verify environment variables
3. Test locally if possible
4. Review this troubleshooting guide
5. Check AWS S3 and IAM permissions

Last Updated: December 13, 2024
Service Version: 1.0.0
Python Version: 3.11.6
Flask Version: 3.0.3