Deployment Guide - Speech Scoring Tool

This guide provides step-by-step instructions for deploying the Speech Scoring Tool both locally and on cloud platforms.

Table of Contents

Local Deployment

Prerequisites

- Python 3.8 or higher
- pip (Python package manager)
- Git

Step 1: Clone Repository

```bash
git clone https://github.com/yourusername/speech-scoring-tool.git
cd speech-scoring-tool
```

Step 2: Create Virtual Environment

```bash
# Create virtual environment
python -m venv venv

# Activate it
# Windows:
venv\Scripts\activate

# macOS/Linux:
source venv/bin/activate
```

Step 3: Install Dependencies

```bash
pip install -r requirements.txt
```

Step 4: Download NLTK Data

```bash
python -c "import nltk; nltk.download('punkt'); nltk.download('brown'); nltk.download('averaged_perceptron_tagger')"
```

Step 5: Run the Application

```bash
# Development mode
python app.py

# Production mode with Gunicorn
gunicorn -w 4 -b 0.0.0.0:5000 app:app
```

Step 6: Test the API

```bash
# Health check
curl http://localhost:5000/api/health

# Test analysis
curl -X POST http://localhost:5000/api/analyze \
  -H "Content-Type: application/json" \
  -d '{
    "transcript": "Hello everyone, myself Muskan...",
    "duration_seconds": 52
  }'
```

Accessing from Network

To access from other devices on your network:

1. Find your local IP: `ipconfig` (Windows) or `ifconfig` (macOS/Linux)
2. Access via: `http://YOUR_IP:5000`
3. Ensure firewall allows port 5000

AWS EC2 Deployment

Step 1: Launch EC2 Instance

1. Sign in to AWS Console
2. Navigate to EC2 Dashboard
3. Click "Launch Instance"
4. Select **Ubuntu Server 22.04 LTS**
5. Choose **t2.micro** (Free Tier eligible)
6. Configure Security Group:
   - SSH (22) - Your IP
   - HTTP (80) - Anywhere
   - Custom TCP (5000) - Anywhere
7. Create/select key pair
8. Launch instance

Step 2: Connect to Instance

```bash
# Make key executable (first time only)
chmod 400 your-key.pem

# Connect
ssh -i "your-key.pem" ubuntu@your-instance-public-ip
```

Step 3: Install Dependencies

```bash
# Update system
sudo apt update
sudo apt upgrade -y

# Install Python and pip
sudo apt install python3 python3-pip python3-venv -y

# Install Git
sudo apt install git -y

# Install Java (required for LanguageTool)
sudo apt install default-jre -y
```

Step 4: Clone and Setup Application

```bash
# Clone repository
git clone https://github.com/yourusername/speech-scoring-tool.git
cd speech-scoring-tool

# Create virtual environment
python3 -m venv venv
source venv/bin/activate

# Install requirements
pip install -r requirements.txt

# Download NLTK data
python3 -c "import nltk; nltk.download('punkt'); nltk.download('brown')"
```

Step 5: Run with Gunicorn

```bash
# Test first
gunicorn -w 4 -b 0.0.0.0:5000 app:app

# For production, use systemd service
sudo nano /etc/systemd/system/speech-scorer.service
```

Service file content:

```ini
[Unit]
Description=Speech Scoring Tool
After=network.target

[Service]
User=ubuntu
WorkingDirectory=/home/ubuntu/speech-scoring-tool
Environment="PATH=/home/ubuntu/speech-scoring-tool/venv/bin"
ExecStart=/home/ubuntu/speech-scoring-tool/venv/bin/gunicorn -w 4 -b 0.0.0.0:5000 app:app

[Install]
WantedBy=multi-user.target
```

```bash
```

```
# Enable and start service
sudo systemctl enable speech-scorer
sudo systemctl start speech-scorer
sudo systemctl status speech-scorer
```

Step 6: Setup Nginx (Optional - for production)

```
bash

# Install Nginx
sudo apt install nginx -y

# Configure
sudo nano /etc/nginx/sites-available/speech-scorer
```

Nginx configuration:

```
nginx

server {
    listen 80;
    server_name your-domain.com;

    location / {
        proxy_pass http://127.0.0.1:5000;
        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;
    }
}
```

```
bash

# Enable site
sudo ln -s /etc/nginx/sites-available/speech-scorer /etc/nginx/sites-enabled/
sudo nginx -t
sudo systemctl restart nginx
```

Heroku Deployment

Step 1: Install Heroku CLI

```
bash
```

```
# macOS
brew tap heroku/brew && brew install heroku

# Windows - Download from heroku.com
# Ubuntu
curl https://cli-assets.heroku.com/install.sh | sh
```

Step 2: Prepare Application

Create `Procfile`:

```bash
echo "web: gunicorn app:app" > Procfile
```

Create `runtime.txt`:

```bash
echo "python-3.11.5" > runtime.txt
```

Step 3: Deploy

```bash
# Login to Heroku
heroku login

# Create app
heroku create speech-scoring-tool

# Add Python buildpack
heroku buildpacks:add heroku/python

# Deploy
git add .
git commit -m "Prepare for Heroku deployment"
git push heroku main

# Open app
heroku open
```

Step 4: View Logs

```bash
```

```
heroku logs --tail
```

Railway Deployment

Step 1: Prepare Repository

Ensure your repository has:

- `requirements.txt`
- `Procfile` (optional)
- `runtime.txt` (optional)

Step 2: Deploy on Railway

1. Go to railway.app
2. Sign in with GitHub
3. Click "New Project"
4. Select "Deploy from GitHub repo"
5. Choose your repository
6. Railway auto-detects Python and deploys
7. Add environment variables if needed
8. Get deployment URL from dashboard

Step 3: Configure

```bash
# Add custom start command (if needed)
# In Railway dashboard → Settings → Deploy
# Start Command: gunicorn -w 4 app:app
```

Docker Deployment

Step 1: Create Dockerfile

```dockerfile
```

```
FROM python:3.11-slim

# Install Java for LanguageTool
RUN apt-get update && apt-get install -y default-jre && rm -rf /var/lib/apt/lists/*

WORKDIR /app

# Copy requirements first for better caching
COPY requirements.txt .
RUN pip install --no-cache-dir -r requirements.txt

# Download NLTK data
RUN python -c "import nltk; nltk.download('punkt'); nltk.download('brown')"

# Copy application
COPY . .

EXPOSE 5000

CMD ["gunicorn", "-w", "4", "-b", "0.0.0.0:5000", "app:app"]
```

Step 2: Create .dockerignore

```
venv/
__pycache__/
*.pyc
.git/
.env
```

Step 3: Build and Run

bash

```
# Build image
docker build -t speech-scoring-tool .

# Run container
docker run -p 5000:5000 speech-scoring-tool

# Or with docker-compose
docker-compose up
```

docker-compose.yml:

yaml

```
version: '3.8'

services:
  app:
    build: .
    ports:
      - "5000:5000"
    environment:
      - FLASK_ENV=production
    restart: unless-stopped
```

Environment Variables

Create `.env` file for local development:

```
bash

FLASK_ENV=development
FLASK_DEBUG=1
PORT=5000
```

For production:

```
bash

FLASK_ENV=production
FLASK_DEBUG=0
PORT=5000
```

Troubleshooting

Port Already in Use

```
bash
```

```
# Find process using port 5000
# Linux/macOS
lsof -i :5000

# Windows
netstat -ano | findstr :5000

# Kill process
kill -9 <PID>
```

LanguageTool Issues

```bash
# Ensure Java is installed
java -version

# If missing, install:
# Ubuntu/Debian
sudo apt install default-jre

# macOS
brew install openjdk
```

NLTK Data Missing

```bash
python -c "import nltk; nltk.download('all')"
```

Memory Issues on Free Tier

```bash
# Reduce Gunicorn workers
gunicorn -w 2 -b 0.0.0.0:5000 app:app

# Or use single worker
gunicorn -w 1 --threads 2 -b 0.0.0.0:5000 app:app
```

Performance Optimization

1. Cache LanguageTool Instance

Already implemented in `app.py` with singleton pattern.

2. Add Redis Caching (Optional)

```python
from flask_caching import Cache

cache = Cache(app, config={'CACHE_TYPE': 'redis'})

@cache.memoize(timeout=300)
def analyze_transcript(transcript):
    # ... analysis code
```

3. Use Production WSGI Server

Always use Gunicorn or uWSGI in production, never Flask's built-in server.

Security Best Practices

1. **Never commit secrets**: Use environment variables
2. **CORS Configuration**: Restrict origins in production
3. **Rate Limiting**: Add Flask-Limiter
4. **HTTPS**: Use SSL certificates (Let's Encrypt)
5. **Input Validation**: Already implemented

Monitoring

Add Logging

```python
import logging

logging.basicConfig(level=logging.INFO)
logger = logging.getLogger(__name__)

logger.info(f"Analyzing transcript of {word_count} words")
```

Health Checks

Already available at `/api/health`

Scaling Considerations

1. **Horizontal Scaling**: Use load balancer + multiple instances
2. **Caching**: Redis for frequent analyses
3. **Async Processing**: Celery for background tasks
4. **Database**: Store results for analytics

Next Steps After Deployment

1. Test all endpoints
2. Monitor logs for errors
3. Set up alerts
4. Configure backups
5. Add analytics
6. Implement user feedback collection

Support

For deployment issues:

1. Check logs first
2. Review error messages
3. Verify all dependencies installed
4. Check firewall/security group settings
5. Open GitHub issue if needed