# Issue #4: Railway.com deployment configuration

**Labels:** `infrastructure`, `deployment`, `production`
**Milestone:** Production Ready
**Estimated Effort:** 6-8 hours

## Description

Configure the pp-bot for deployment on Railway.com, including PostgreSQL database integration, environment variable management, health checks, and automatic deployments from GitHub.

## Prerequisites

- [ ] Issue #2 (PostgreSQL integration) must be completed first
- [ ] Railway.com account created
- [ ] GitHub repository connected to Railway

## Tasks

### Phase 1: Railway Project Setup (1 hour)

- [ ] Create new project on Railway.com
- [ ] Connect GitHub repository (stevencarpenter/pp-bot)
- [ ] Configure automatic deployments from `main` branch
- [ ] Set up PostgreSQL database addon
- [ ] Copy `DATABASE_URL` from Railway to environment variables

### Phase 2: Application Configuration (2-3 hours)

- [ ] Create `railway.toml` or `railway.json` configuration
- [ ] Configure build command: `npm run build` (if TypeScript)
- [ ] Configure start command: `npm start`
- [ ] Set up health check endpoint
- [ ] Configure restart policy
- [ ] Set resource limits (free/hobby tier optimization)

### Phase 3: Environment Variables (1 hour)

- [ ] Transfer all environment variables to Railway
- `SLACK_BOT_TOKEN`
- `SLACK_APP_TOKEN`
- `SLACK_SIGNING_SECRET`
- `DATABASE_URL` (auto-provided by Railway)
- `NODE_ENV=production`
- `PORT` (auto-provided by Railway)
- [ ] Verify no secrets in repository

- [ ] Document environment variable setup process

### Phase 4: Database Initialization (1-2 hours)

- [ ] Run database migrations on Railway PostgreSQL
- [ ] Test database connectivity from Railway app
- [ ] Migrate existing data (if any)
- [ ] Set up automated backups (Railway feature)

### Phase 5: Testing & Validation (2-3 hours)

- [ ] Deploy to Railway and verify startup
- [ ] Test health check endpoint
- [ ] Verify bot connects to Slack
- [ ] Test voting functionality
- [ ] Test slash commands
- [ ] Check logs for errors
- [ ] Monitor resource usage
- [ ] Test automatic redeployment on push

## Railway Configuration

### railway.toml

```toml
[build]
builder = "NIXPACKS"
buildCommand = "npm ci && npm run build"

[deploy]
startCommand = "npm start"
healthcheckPath = "/health"
healthcheckTimeout = 100
restartPolicyType = "ON_FAILURE"
restartPolicyMaxRetries = 10

[env]
NODE_ENV = "production"
```

### Alternative: railway.json

```json
{
  "build": {
    "builder": "NIXPACKS",
    "buildCommand": "npm ci && npm run build"
  },
  "deploy": {
    "startCommand": "npm start",
    "healthcheckPath": "/health",
    "healthcheckTimeout": 100,
    "restartPolicyType": "ON_FAILURE",
    "restartPolicyMaxRetries": 10
  }
}
```

# Health Check Endpoint

## Add to Application Code

```
// health.js
const express = require('express');

function createHealthServer(port = 3001) {
  const app = express();

  let isReady = false;
  let dbConnected = false;

  // Liveness probe - is the app running?
  app.get('/health', (req, res) => {
    res.status(200).json({
      status: 'ok',
      timestamp: new Date().toISOString(),
      uptime: process.uptime(),
    });
  });

  // Readiness probe - is the app ready to serve?
  app.get('/ready', (req, res) => {
    if (isReady && dbConnected) {
      res.status(200).json({
        status: 'ready',
        database: 'connected',
        slack: 'connected',
      });
    } else {
      res.status(503).json({
        status: 'not ready',
        database: dbConnected ? 'connected' : 'disconnected',
        slack: isReady ? 'connected' : 'disconnected',
      });
    }
  });

  const server = app.listen(port, () => {
    console.log(`Health check server running on port ${port}`);
  });

  return {
    server,
    setReady: (ready) => { isReady = ready; },
    setDbConnected: (connected) => { dbConnected = connected; },
  };
}

module.exports = createHealthServer;
```

### Integration in Main App

```
// index.js
const createHealthServer = require('./health');
const pool = require('./db');

// Start health check server
const healthServer = createHealthServer(process.env.HEALTH_PORT || 3001);

// Check database connection
pool.query('SELECT 1')
  .then(() => {
    console.log('✓ Database connected');
    healthServer.setDbConnected(true);
  })
  .catch(err => {
    console.error('✗ Database connection failed:', err);
    healthServer.setDbConnected(false);
  });

// Start Slack bot
app.start(process.env.PORT || 3000)
  .then(() => {
    console.log('⚡ Bolt app is running!');
    healthServer.setReady(true);
  })
  .catch(err => {
    console.error('✗ Failed to start app:', err);
    process.exit(1);
  });

// Graceful shutdown
process.on('SIGTERM', async () => {
  console.log('SIGTERM received, shutting down gracefully...');
  healthServer.setReady(false);

  await app.stop();
  await pool.end();
  healthServer.server.close();

  process.exit(0);
});
```

## Package.json Updates

```
{
  "scripts": {
    "start": "node dist/index.js",
    "build": "tsc",
    "deploy": "railway up",
    "logs": "railway logs",
    "db:migrate": "railway run npm run migrate"
  },
  "engines": {
    "node": ">=18.0.0",
    "npm": ">=9.0.0"
  }
}
```

# Railway CLI Commands

## Useful Commands for Development

```
# Install Railway CLI
npm i -g @railway/cli

# Login to Railway
railway login

# Link to project
railway link

# Run commands with Railway environment
railway run npm start

# View logs
railway logs

# Open Railway dashboard
railway open

# Deploy manually
railway up
```

# Database Migration on Railway

## Run Migrations

```
# Option 1: Via Railway CLI
railway run node scripts/migrate.js

# Option 2: Via Railway dashboard
# Navigate to project → PostgreSQL → Query
# Paste and execute migration SQL

# Option 3: Automated in start script
# Add to package.json:
"start": "npm run migrate && node dist/index.js"
```

### Migration Script

```
// scripts/migrate.js
const pool = require('../db');
const fs = require('fs');
const path = require('path');

async function runMigrations() {
  try {
    const migrationFile = path.join(__dirname, '..', 'migrations', '001_initial_schema
.sql');
    const sql = fs.readFileSync(migrationFile, 'utf8');

    await pool.query(sql);
    console.log('✓ Migrations completed successfully');

    await pool.end();
    process.exit(0);
  } catch (error) {
    console.error('✗ Migration failed:', error);
    process.exit(1);
  }
}

runMigrations();
```

## Free Tier Optimization

### Resource Limits

Railway.com free tier includes:
- $5 credit per month
- ~512 MB RAM
- Shared CPU
- 1 GB disk storage

### Optimization Tips

```
// Optimize connection pool for free tier
const pool = new Pool({
  connectionString: process.env.DATABASE_URL,
  max: 5, // Reduced from 10
  idleTimeoutMillis: 30000,
  connectionTimeoutMillis: 2000,
});

// Add memory monitoring
function logMemoryUsage() {
  const usage = process.memoryUsage();
  console.log({
    rss: `${Math.round(usage.rss / 1024 / 1024)}MB`,
    heapUsed: `${Math.round(usage.heapUsed / 1024 / 1024)}MB`,
    heapTotal: `${Math.round(usage.heapTotal / 1024 / 1024)}MB`,
  });
}

setInterval(logMemoryUsage, 60000); // Log every minute
```

# Monitoring & Logging

## Structured Logging

```
const winston = require('winston');

const logger = winston.createLogger({
  level: process.env.LOG_LEVEL || 'info',
  format: winston.format.combine(
    winston.format.timestamp(),
    winston.format.json()
  ),
  transports: [
    new winston.transports.Console({
      format: winston.format.simple(),
    }),
  ],
});

module.exports = logger;
```

## Usage

```
const logger = require('./logger');

logger.info('Vote processed', {
  userId: 'U12345',
  action: '++',
  newScore: 5
});

logger.error('Database error', {
  error: err.message,
  stack: err.stack
});
```

# Acceptance Criteria

- [ ] Application deploys successfully to Railway.com
- [ ] PostgreSQL database is connected and working
- [ ] All environment variables configured correctly
- [ ] Health check endpoint responds correctly
- [ ] Bot connects to Slack successfully
- [ ] All bot functionality works in production
- [ ] Automatic deployments work on push to main
- [ ] Logs are accessible and useful
- [ ] Resource usage within free tier limits
- [ ] Documentation complete for deployment process

# Testing Checklist

- [ ] Health check returns 200 OK
- [ ] `/ready` endpoint shows all systems connected

- [ ] Vote processing works
- [ ] Leaderboard command works
- [ ] Database persists data across restarts
- [ ] Automatic redeployment on git push
- [ ] Error handling works correctly
- [ ] Graceful shutdown on SIGTERM

## Rollback Plan

If deployment fails:
1. Railway automatically keeps previous deployment running
2. Use Railway dashboard to rollback to previous version
3. Check logs for error details
4. Fix issues locally and redeploy
5. If critical, switch back to local/development setup

## Documentation Requirements

- [ ] Update README.md with deployment instructions
- [ ] Create DEPLOYMENT.md with detailed Railway setup
- [ ] Document environment variables
- [ ] Add troubleshooting guide
- [ ] Include Railway CLI commands
- [ ] Add monitoring/logging information

## Resources

- Railway.com Documentation (https://docs.railway.app/)
- Railway Node.js Guide (https://docs.railway.app/guides/nodejs)
- Railway PostgreSQL Guide (https://docs.railway.app/databases/postgresql)
- Railway CLI Reference (https://docs.railway.app/develop/cli)

## Dependencies

**Requires:**
- Issue #2: PostgreSQL integration (MUST be completed first)

**Blocks:**
- Issue #5: CI/CD workflow (deployment target)