

Azure App Service Deployment Guide

React + Node.js Application

Node.js 24 LTS Runtime
GitHub Actions CI/CD
Azure SQL Database

Template Version: 1.0
Last Updated: February 2026

Azure App Service Deployment Guide - React + Node.js

Runtime: Node.js 24 LTS
Deployment Method: GitHub Actions
Database: Azure SQL Database

Summary

This document provides a complete step-by-step guide for deploying a React + Node.js application to Azure App Service with Node 24 LTS runtime, including common issues and solutions.

Prerequisites

- Azure subscription
- GitHub account
- Node.js 24 LTS installed locally
- Azure SQL Database instance (if applicable)
- Git repository

Project Structure

```
your-app-name/
% % % dist/                # Built React frontend (generated)
% % % server/              # Node.js Express backend
% % %   % % % index.js      # Main server entry point
% % %   % % % .env          # Environment variables
% % %   % % % package.json  # Server dependencies
% % %   % % % src/          # React source code
% % %   % % % package.json  # Root package.json
% % %   % % % vite.config.js # Vite build configuration
% % %   % % % web.config     # IIS configuration for Azure
% % %   % % % deploy.cmd     # Custom deployment script
% % %   % % % .deployment    # Deployment configuration
% % %   % % % .github/
% % %       % % % workflows/
% % %           % % % main_your-app.yml # GitHub Actions workflow
```

Step 1: Prepare Application for Node 24 LTS

1.1 Update Package.json Engine Requirements

Root package.json:

```
{
  "engines": {
    "node": ">=20.0.0",
    "npm": ">=10.0.0"
  }
}
```

server/package.json:


```
{
  "engines": {
    "node": ">=20.0.0",
    "npm": ">=10.0.0"
  }
}
```

1.2 Handle Platform-Specific Dependencies

Issue: Platform-specific binaries (like Windows-only packages) cause build failures on Linux build agents.

Solution: Move platform-specific dependencies to optionalDependencies in package.json

```
{
  "devDependencies": {
    "vite": "^6.0.5"
  },
  "optionalDependencies": {
    "@rollup/rollup-win32-x64-msvc": "^4.55.2"
  }
}
```

Step 2: Configure Azure App Service

2.1 Create App Service

Azure Portal:

1. Create Resource - Web App
2. Settings:
 - **Name:** your-app-name
 - **Runtime Stack:** Node 24 LTS
 - **Operating System:** Linux
 - **Region:** Your preferred region
 - **Pricing Plan:** B1 or higher

2.2 Configure Application Settings

Azure Portal - Configuration - Application settings:

Add these environment variables (adjust to your needs):

```
NODE_ENV=production
DB_SERVER=your-db-server.database.windows.net
DB_NAME=your-database-name
DB_USER=your-db-user
DB_PASSWORD=your-db-password
DB_ENCRYPT=yes
DB_TRUST_SERVER_CERTIFICATE=no
DB_PORT=1433
JWT_SECRET=your-jwt-secret-here
JWT_EXPIRES_IN=1h
CSRF_SECRET=your-csrf-secret-here
SESSION_SECRET=your-session-secret-here
SESSION_DURATION_HOURS=8
SMTP_HOST=smtp.gmail.com
SMTP_PORT=587
SMTP_SECURE=false
SMTP_USER=your-email@domain.com
SMTP_PASS=your-email-app-password
ALLOWED_ORIGINS=https://your-app-name.azurewebsites.net
```

Critical: NODE_ENV=production is required for the server to serve the React frontend.

2.3 Configure Startup Command

Azure Portal - Configuration - General settings:

Startup Command:

node server/index.js

Step 3: Setup GitHub Actions Deployment

3.1 Configure Deployment Center

Azure Portal - Deployment Center

1. Source: GitHub
2. Organization: Your-Organization
3. Repository: your-repo-name
4. Branch: main
5. Authentication: User-assigned identity

This automatically creates the GitHub Actions workflow file.

3.2 Update GitHub Actions Workflow

File: ``.github/workflows/main_your-app.yml``

```
name: Build and deploy Node.js app to Azure Web App
on:
  push:
    branches:
      - main
  workflow_dispatch:
jobs:
  build:
    runs-on: ubuntu-latest
    permissions:
      contents: read
    steps:
      - uses: actions/checkout@v4
      - name: Set up Node.js version
        uses: actions/setup-node@v3
        with:
          node-version: '24.x'
      - name: npm install, build, and test
        run: |
          npm install
          npm run build
          npm run test --if-present
      - name: Install server dependencies
        run: |
          cd server
          npm install --production
      - name: Verify build artifacts
        run: |
          echo "Checking dist folder..."
          ls -la dist/
          echo "Checking server folder..."
          ls -la server/
      - name: Upload artifact for deployment job
        uses: actions/upload-artifact@v4
        with:
          name: node-app
          path: .
  deploy:
    runs-on: ubuntu-latest
    needs: build
    permissions:
      id-token: write
      contents: read
    steps:
      - name: Download artifact from build job
        uses: actions/download-artifact@v4
        with:
          name: node-app
```

```

- name: Login to Azure
  uses: azure/login@v2
  with:
    client-id: ${ secrets.AZUREAPPSERVICE_CLIENTID }
    tenant-id: ${ secrets.AZUREAPPSERVICE_TENANTID }
    subscription-id: ${ secrets.AZUREAPPSERVICE_SUBSCRIPTIONID }

- name: 'Deploy to Azure Web App'
  uses: azure/webapps-deploy@v3
  with:
    app-name: 'your-app-name'
    slot-name: 'Production'
    package: .

```

Key additions:

- Install server dependencies separately
- Verify build artifacts before upload
- Changed `npm run build --if-present` to `npm run build` (mandatory)

Step 4: Configure Server for Production

4.1 Update server/index.js

Critical code for serving React app:

```

const PORT = process.env.PORT || process.env.WEBSITES_PORT || 8080;

// Serve static frontend files in production
if (process.env.NODE_ENV === 'production') {
  const distPath = path.join(__dirname, '..', 'dist');
  const fs = require('fs');

  console.log(`NODE_ENV: ${process.env.NODE_ENV}`);
  console.log(`Looking for dist folder at: ${distPath}`);
  console.log(`Dist folder exists: ${fs.existsSync(distPath)}`);

  if (fs.existsSync(distPath)) {
    console.log(`Dist folder contents:`, fs.readdirSync(distPath));
  }

  app.use(express.static(distPath, {
    setHeaders: (res, filePath) => {
      if (filePath.endsWith('.js')) {
        res.setHeader('Content-Type', 'application/javascript');
      } else if (filePath.endsWith('.css')) {
        res.setHeader('Content-Type', 'text/css');
      } else if (filePath.endsWith('.html')) {
        res.setHeader('Content-Type', 'text/html');
      }
    }
  }));
  console.log(`Serving static files from: ${distPath}`);
}

// Serve React app for all non-API routes
if (process.env.NODE_ENV === 'production') {
  app.get('*', (req, res) => {
    if (req.path.startsWith('/api') || req.path.startsWith('/health')) {
      return res.status(404).json({ error: 'Route not found' });
    }
    res.sendFile(path.join(__dirname, '..', 'dist', 'index.html'));
  });
}

```

4.2 Configure CORS for Production


```

const allowedOrigins = [
  'http://localhost:5173',
  'https://your-app-name.azurewebsites.net',
  process.env.CORS_ORIGIN
].filter(Boolean);
app.use(cors({
  origin: function (origin, callback) {
    if (!origin) {
      return callback(null, true);
    }
    if (allowedOrigins.includes(origin)) {
      callback(null, true);
    } else if (process.env.NODE_ENV === 'production') {
      const requestHost = origin.replace(/^https?:\/\//, '');
      const serverHost = process.env.WEBSITE_HOSTNAME || origin;
      if (requestHost === serverHost) {
        callback(null, true);
      } else {
        callback(new Error('Not allowed by CORS'));
      }
    }
  },
  credentials: true
}));

```

Step 5: Database Configuration

5.1 Update Database Connection

File: `server/config/database.js`

```

// Support Windows Authentication (local) and SQL Auth (Azure)
const requiredEnvVars = ['DB_SERVER', 'DB_NAME'];
if (process.env.DB_INTEGRATED_SECURITY !== 'true') {
  requiredEnvVars.push('DB_USER', 'DB_PASSWORD');
}
const config = {
  server: process.env.DB_SERVER,
  database: process.env.DB_NAME,
  port: parseInt(process.env.DB_PORT || '1433'),
  options: {
    encrypt: process.env.DB_ENCRYPT === 'yes',
    trustServerCertificate: process.env.DB_TRUST_SERVER_CERTIFICATE === 'yes',
    enableArithAbort: true,
  },
  pool: {
    max: parseInt(process.env.DB_POOL_MAX || '10'),
    min: parseInt(process.env.DB_POOL_MIN || '0'),
  },
};
// Add authentication based on configuration
if (process.env.DB_INTEGRATED_SECURITY === 'true') {
  config.options.trustedConnection = true;
} else {
  config.user = process.env.DB_USER;
  config.password = process.env.DB_PASSWORD;
}

```

5.2 Azure SQL Firewall

Azure Portal - SQL Database - Networking:

- Enable "Allow Azure services and resources to access this server"
- Add your IP if you need local access

Common Issues and Solutions

Issue 1: "Application Error" on Deployment

Symptom: Blank page with "Application Error" message

Cause: Multiple factors:

1. Missing NODE_ENV=production environment variable
2. Server dependencies not installed during build
3. dist folder not being created/uploaded

Solution:

1. Set NODE_ENV=production in Azure App Settings
2. Add server dependency installation step in workflow
3. Change `npm run build --if-present` to `npm run build` (mandatory)

Issue 2: API JSON Response Instead of React App

Symptom: Visiting the URL shows API JSON response instead of the React UI

Cause: NODE_ENV was not set to production, so the server didn't serve static files

Solution: Set NODE_ENV=production in Azure Application Settings

Issue 3: Platform-Specific Dependency Build Failure

Symptom:

```
npm error notsup Unsupported platform for package
npm error notsup Valid os: win32
npm error notsup Actual os: linux
```

Cause: Windows-specific binary in devDependencies on Linux build agent

Solution: Move to optionalDependencies

```
"optionalDependencies": {
  "@rollup/rollup-win32-x64-msvc": "^4.55.2"
}
```

Issue 4: Missing Environment Variables

Symptom: Database connection errors in Azure logs

Cause: Environment variables only existed in local .env file

Solution: Add all required environment variables to Azure App Service Configuration

Issue 5: Port Binding Issues Locally

Symptom: EADDRINUSE: address already in use

Cause: Multiple node processes running on the same port

Solution:

```
Get-Process -Name node | Stop-Process -Force
```

Or change port in server/.env

```
PORT=5001
```

Deployment Checklist

Pre-Deployment

- [] Node 24+ specified in package.json engines
- [] Platform-specific dependencies in optionalDependencies
- [] npm run build creates dist folder successfully
- [] server/.env configured for local testing
- [] Database connection tested locally

Azure Configuration

- [] App Service created with Node 24 LTS runtime
- [] All environment variables added to Application Settings
- [] NODE_ENV=production set in Application Settings
- [] Startup command: node server/index.js
- [] Azure SQL firewall allows Azure services (if using database)

GitHub Actions

- [] Workflow file includes server dependency installation
- [] Build step runs npm run build (not --if-present)
- [] Artifact upload includes entire project directory
- [] Azure credentials configured in GitHub secrets

Post-Deployment Verification

- [] GitHub Actions workflow completes successfully
- [] Azure Log Stream shows successful startup
- [] Visiting app URL loads React frontend (not API JSON)
- [] API endpoints respond correctly
- [] Database connection successful (if applicable)

Testing the Deployment

1. Health Check

```
curl https://your-app-name.azurewebsites.net/health
```

Should return API status JSON.

2. Frontend

Visit <https://your-app-name.azurewebsites.net> in browser

Should load the React application.

3. View Logs

Azure Portal - App Service - Log stream

Look for:

```
NODE_ENV: production
Looking for dist folder at: /home/site/wwwroot/dist
Dist folder exists: true
Serving static files from: /home/site/wwwroot/dist
Server started on port 8080
Database connection pool created successfully
```

Monitoring and Troubleshooting

View Live Logs

```
az webapp log tail --name your-app-name --resource-group your-resource-group
```

Common Issues

App shows API JSON instead of React:

- Check NODE_ENV=production in App Settings
- Verify dist folder exists in deployment
- Check Log Stream for static file serving messages

Database connection errors:

- Verify all DB_* environment variables in App Settings
- Check Azure SQL firewall rules

- Verify connection string format

Build failures:ð

- Check GitHub Actions logs
- Verify Node version matches (24.x)
- Check for platform-specific dependencies

404 on routes:ð

- Ensure catch-all route serves index.html
- Check CORS configuration
- Verify API routes are excluded from catch-all

Local Development vs Productionð

Local Developmentð

- PORT: 5001 (frontend: 5173)
- NODE_ENV: development
- Database: Can use Windows Authentication or SQL Auth
- CORS: localhost origins

Production (Azure)ð

- PORT: 8080 (set by Azure)
- NODE_ENV: production
- Database: SQL Authentication required
- CORS: Azure App Service URL
- Static files served from dist/

Useful Commandsð

Local Developmentð

```
# Install dependenciesð
npm installð
cd server && npm installð
ð
# Build frontendð
npm run buildð
ð
# Run server locallyð
npm run serverð
ð
# Run frontend dev serverð
npm run devð
```

Azure CLIð

```
# View logsð
az webapp log tail --name your-app-name --resource-group your-rgð
ð
# Restart appð
az webapp restart --name your-app-name --resource-group your-rgð
ð
# View configð
az webapp config appsettings list --name your-app-name --resource-group your-rgð
ð
# Set environment variableð
az webapp config appsettings set --name your-app-name --resource-group your-rg --settings KEY=VALUEð
```

Gitð


```
# Commit and deploy
git add .
git commit -m "message"
git push origin main
```

Success Criteria

- GitHub Actions workflow completes without errors
- Azure App Service status shows "Running"
- Visiting the URL loads the React frontend
- API endpoints accessible at /api/*
- Database connection successful (if applicable)
- No errors in Azure Log Stream
- Health check endpoint returns status

Key Lessons

1. **NODE_ENV is critical** - Without NODE_ENV=production, the server won't serve static files
2. **Platform-specific dependencies** - Use optionalDependencies for platform-specific binaries
3. **Build must run** - Change --if-present to mandatory build
4. **Server dependencies** - Must be installed separately in the workflow
5. **Environment variables** - All must be configured in Azure App Settings
6. **Debugging** - Add console.log statements to verify dist folder and configuration
7. **Artifact uploads** - Include entire directory (.) to get all files including dist

Document Template Version: 1.0

Last Updated: February 2026

Recommended Runtime: Node.js 24 LTS

