Replit App Deployment Automation Guide

Option 1: Replit + Railway/Render (Recommended)

Why This Approach:

- **Easiest setup**: Railway and Render have excellent Replit integration
- **Automatic deployments**: Push to GitHub → Auto-deploy
- **Database included**: PostgreSQL/MySQL automatically provisioned
- **Custom domains**: Easy DNS configuration

Setup Steps:

1. Replit Configuration

Create a `replit.nix` file:

```nix
{ pkgs }: {
  deps = [
    pkgs.nodejs-18_x
    pkgs.postgresql
  ];
}
```

Create a `.replit` file:

```toml
modules = ["nodejs-18"]
run = "npm start"

[deployment]
run = ["sh", "-c", "npm start"]
```

2. Railway Deployment Script

Add this to your `package.json`:

```json
{
  "scripts": {
    "build": "npm install",
    "start": "node server.js",
    "deploy": "railway login && railway up"
  }
}
```

3. Environment Variables

Create `.env` template:

```bash
DATABASE_URL=postgresql://user:password@host:port/database
PORT=3000
DOMAIN=yourdomain.com
```

4. Auto-deployment Script

Create `deploy.sh`:

```bash
#!/bin/bash
echo "🚀 Starting automated deployment..."

# Install Railway CLI
npm install -g @railway/cli

# Login to Railway (use token for automation)
railway login --token $RAILWAY_TOKEN

# Deploy with custom domain
railway up --service $SERVICE_NAME

# Configure custom domain
railway domain add $DOMAIN_NAME

echo "✅ Deployment complete!"
```

Option 2: Replit + Vercel + PlanetScale

Setup Script

```bash
#!/bin/bash
# Install Vercel CLI
npm i -g vercel

# Install PlanetScale CLI
curl -fsSL https://github.com/planetscale/cli/releases/latest/download/pscale_linux_amd64.tar.gz | tar -xz pscale
sudo mv pscale /usr/local/bin

# Deploy
vercel --prod
pscale database create $DB_NAME
pscale connect $DB_NAME main --port 3309
```

Option 3: Full Docker Automation

Dockerfile

```dockerfile
FROM node:18-alpine
WORKDIR /app
COPY package*.json ./
RUN npm install
COPY . .
EXPOSE 3000
CMD ["npm", "start"]
```

docker-compose.yml

```yaml
version: '3.8'
services:
 app:
  build: .
  ports:
   - "3000:3000"
  environment:
   - DATABASE_URL=postgresql://postgres:password@db:5432/myapp
  depends_on:
   - db

 db:
  image: postgres:14
  environment:
   - POSTGRES_DB=myapp
   - POSTGRES_PASSWORD=password
  volumes:
   - postgres_data:/var/lib/postgresql/data

volumes:
 postgres_data:
```

Deployment Script

```bash
#!/bin/bash
# Build and deploy
docker-compose up -d

# Configure nginx reverse proxy
sudo nginx -t && sudo systemctl reload nginx
```

Recommended: One-Click Deployment Template

create-deployment.js

```javascript
const { exec } = require('child_process');
const fs = require('fs');

const deployApp = async (config) => {
  const { appName, domain, dbType = 'postgresql' } = config;

  console.log(`🚀 Deploying ${appName} to ${domain}...`);

  // 1. Create Railway project
  exec(`railway create ${appName}`, (error, stdout) => {
    if (error) throw error;
    console.log('✅ Railway project created');
  });

  // 2. Add database
  exec(`railway add ${dbType}`, (error, stdout) => {
    if (error) throw error;
    console.log('✅ Database added');
  });

  // 3. Deploy code
  exec('railway up', (error, stdout) => {
    if (error) throw error;
    console.log('✅ Code deployed');
  });

  // 4. Configure custom domain
  exec(`railway domain add ${domain}`, (error, stdout) => {
    if (error) throw error;
    console.log(`✅ Domain ${domain} configured`);
  });
};

// Usage
deployApp({
  appName: 'my-awesome-app',
  domain: 'myapp.com',
  dbType: 'postgresql'
});
```

DNS Configuration

Cloudflare Setup (Recommended)

```bash
# Install Cloudflare CLI
npm install -g @cloudflare/cli

# Configure DNS records
curl -X POST "https://api.cloudflare.com/client/v4/zones/YOUR_ZONE_ID/dns_records" \
 -H "Authorization: Bearer YOUR_API_TOKEN" \
 -H "Content-Type: application/json" \
 --data '{
  "type": "CNAME",
  "name": "your-subdomain",
  "content": "your-app.railway.app"
 }'
```

Environment Variables Management

.env.example

```bash
# Database
DATABASE_URL=
DB_HOST=
DB_PORT=
DB_NAME=
DB_USER=
DB_PASSWORD=

# App
PORT=3000
NODE_ENV=production
JWT_SECRET=

# Domain
DOMAIN=
SUBDOMAIN=

# API Keys
CLOUDFLARE_API_TOKEN=
RAILWAY_TOKEN=
```

Complete Automation Script

auto-deploy.sh

```bash
#!/bin/bash
set -e

echo "🎯 Automated App Deployment Starting..."

# Configuration
APP_NAME=$1
DOMAIN=$2
DB_TYPE=${3:-postgresql}

if [ -z "$APP_NAME" ] || [ -z "$DOMAIN" ]; then
  echo "Usage: ./auto-deploy.sh <app-name> <domain.com> [db-type]"
  exit 1
fi

# 1. Setup Railway
echo "📦 Setting up Railway..."
railway login --token $RAILWAY_TOKEN
railway create $APP_NAME

# 2. Add database
echo "🗄 Adding database..."
railway add $DB_TYPE

# 3. Deploy application
echo "🚀 Deploying application..."
railway up

# 4. Configure domain
echo "🌐 Configuring domain..."
railway domain add $DOMAIN

# 5. Setup SSL
echo "🔒 Setting up SSL..."
# Railway handles SSL automatically

echo "✅ Deployment complete!"
echo "🌍 Your app is live at: https://$DOMAIN"
```

Usage

1. **Make the script executable:**

```bash
chmod +x auto-deploy.sh
```

2. **Run deployment:**

```bash
./auto-deploy.sh my-app example.com postgresql
```

Best Practices

1. **Use environment variables** for all sensitive data
2. **Implement health checks** in your application
3. **Set up monitoring** with Railway/Render dashboards
4. **Configure automatic backups** for your database
5. **Use CDN** for static assets (Cloudflare)
6. **Implement CI/CD** with GitHub Actions

Troubleshooting

Common Issues:

- **DNS propagation**: Wait 24-48 hours for full propagation
- **SSL certificates**: Most platforms auto-generate, but may take a few minutes
- **Database connections**: Ensure your app uses the correct DATABASE_URL format
- **Port configuration**: Most platforms auto-assign ports, use `process.env.PORT`