🚂 SOaC Framework - Railway.app Deployment Guide

Deploy your SOaC Framework to Railway.app in under 10 minutes with this comprehensive guide!

📋 Table of Contents

🎯 Why Railway.app?

✅ **Free Tier**: $5 free credit monthly (no credit card required initially)
✅ **Zero Config**: Automatic HTTPS, environment variables, and database setup
✅ **GitHub Integration**: Auto-deploy on push
✅ **PostgreSQL Built-in**: One-click database provisioning
✅ **Easy Scaling**: Upgrade resources as needed
✅ **No Local Docker**: Deploy directly from GitHub

📦 Prerequisites

1. **GitHub Account** - To connect your repository
2. **Railway Account** - Sign up at railway.app (https://railway.app) (free)
3. **SOaC Framework Repository** - Fork or clone this repo to your GitHub

No credit card required for initial deployment!

⚡ Quick Start (3 Steps)

Step 1: Fork the Repository

```
# Fork this repository to your GitHub account via GitHub UI
# or clone and push to your own repo
git clone https://github.com/yourusername/soac-framework.git
cd soac-framework
git remote set-url origin https://github.com/YOUR_USERNAME/soac-framework.git
git push -u origin main
```

Step 2: Deploy to Railway

1. Go to railway.app (https://railway.app)
2. Click **“Start a New Project”**
3. Select **“Deploy from GitHub repo”**
4. Select your `soac-framework` repository
5. Railway will auto-detect and start deployment!

Step 3: Add PostgreSQL Database

1. In your Railway project, click **”+ New”**
2. Select **“Database”** → **“PostgreSQL”**
3. Railway will automatically:
   - Create a PostgreSQL instance
   - Generate a `DATABASE_URL` environment variable
   - Connect it to your backend service

That’s it! Your SOaC Framework is deploying! 🎉

📖 Detailed Step-by-Step Guide

Part 1: Prepare Your Repository

Option A: Using Existing Repository

If you already have the SOaC Framework in your GitHub account, skip to Part 2.

Option B: Fork from Source

1. Go to the SOaC Framework repository on GitHub
2. Click the **“Fork”** button in the top right
3. This creates a copy in your GitHub account

Option C: Push to Your Own Repository

```
# Clone the repository
git clone https://github.com/original-repo/soac-framework.git
cd soac-framework

# Create a new repository on GitHub (via web UI)
# Then set your repo as the remote
git remote set-url origin https://github.com/YOUR_USERNAME/soac-framework.git
git push -u origin main
```

Part 2: Create Railway Project

1. Sign Up for Railway

- Visit railway.app (https://railway.app)
- Click **“Login with GitHub”**
- Authorize Railway to access your GitHub repositories

2. Create a New Project

1. From Railway dashboard, click **”+ New Project”**
2. Select **“Deploy from GitHub repo”**
3. Choose your `soac-framework` repository
4. Railway will analyze your repo and detect:
   - Backend Dockerfile: `backend/Dockerfile.railway`
   - Frontend Dockerfile: `frontend/Dockerfile.railway`

3. Configure Services

Railway will create services automatically, but you need to configure them:

Backend Service Configuration:
1. Click on the backend service
2. Go to **“Settings”** tab
3. Set **Root Directory** to: `backend`
4. Set **Dockerfile Path** to: `Dockerfile.railway`
5. Click **“Save”**

Frontend Service Configuration:
1. Click on the frontend service (or create a new service if not auto-created)
2. Go to **“Settings”** tab
3. Set **Root Directory** to: `frontend`
4. Set **Dockerfile Path** to: `Dockerfile.railway`
5. Click **“Save”**

Part 3: Add PostgreSQL Database

1. Add Database Service

1. In your Railway project, click **”+ New”**
2. Select **“Database”**
3. Choose **“PostgreSQL”**
4. Railway will provision a database and generate connection details

2. Connect Database to Backend

Railway automatically creates a `DATABASE_URL` environment variable that contains:

```
postgresql://user:password@host:port/database
```

This is automatically available to your backend service! No manual configuration needed.

Part 4: Configure Environment Variables

Backend Environment Variables

Click on your **backend service** → **“Variables”** tab and add:

Required Variables:

```
# JWT Secret (GENERATE A NEW ONE!)
SECRET_KEY=your-super-secret-jwt-key-minimum-32-characters-random-string-here

# JWT Configuration
ALGORITHM=HS256
ACCESS_TOKEN_EXPIRE_MINUTES=1440

# Environment
ENVIRONMENT=production

# Mock Mode (set to false when ready for real devices)
MOCK_MODE=true

# Background Event Collection
ENABLE_BACKGROUND_COLLECTION=true
EVENT_COLLECTION_INTERVAL=300
```

After Backend Deploys:
1. Copy the backend service URL (e.g., `https://soac-backend-production.up.railway.app`)
2. Go to frontend service variables and add:

Frontend Environment Variables

```
# Backend API URL (replace with your backend URL from above)
VITE_API_BASE_URL=https://soac-backend-production.up.railway.app
```

1. Then go back to backend variables and add:

```
# Frontend URL (replace with your frontend URL)
FRONTEND_URL=https://soac-frontend-production.up.railway.app
```

💡 **Pro Tip:** To generate a secure SECRET_KEY:

```
# On Linux/Mac:
openssl rand -hex 32

# Or use Python:
python -c "import secrets; print(secrets.token_hex(32))"

# Or use Node.js:
node -e "console.log(require('crypto').randomBytes(32).toString('hex'))"
```

Part 5: Deploy and Access

1. Trigger Deployment

After configuring environment variables:
1. Railway will automatically redeploy both services
2. Monitor deployment logs in each service's **"Deployments"** tab
3. Wait for both services to show **"Success"** status (typically 2-5 minutes)

2. Get Your Application URLs

1. Click on **frontend service**
2. Go to **"Settings"** → **"Networking"**
3. Click **"Generate Domain"**
4. Railway will provide a URL like: `https://soac-frontend-production.up.railway.app`
5. Repeat for **backend service**:
6. You'll get: `https://soac-backend-production.up.railway.app`

3. Access Your Application

Open your frontend URL in a browser:

```
https://soac-frontend-production.up.railway.app
```

Default Login Credentials:
- **Admin**:
- Username: `admin`
- Password: `admin123`
- **Analyst**:
- Username: `analyst`
- Password: `analyst123`

⚠️ **IMPORTANT:** Change these passwords immediately after first login!

🔐 Environment Variables Reference

Backend Service Variables

Required Variables

Variable	Description	Example
`DATABASE_URL`	PostgreSQL connection string	Auto-provided by Railway
`SECRET_KEY`	JWT secret key (generate strong random string)	`your-secret-key-here`
`ALGORITHM`	JWT algorithm	`HS256`
`ACCESS_TOKEN_EXPIRE_MINUTES`	Token expiration time	`1440` (24 hours)
`FRONTEND_URL`	Frontend service URL	`https://your-frontend.railway.app`
`ENVIRONMENT`	Environment name	`production`

Optional Variables

Variable	Description	Default	Example
`MOCK_MODE`	Use mock device data	`true`	`false` (for real devices)
`ENABLE_BACKGROUND_COLLECTION`	Auto-collect events	`true`	`true`
`EVENT_COLLECTION_INTERVAL`	Collection interval (seconds)	`300`	`600`
`LOG_LEVEL`	Logging level	`INFO`	`DEBUG`
`RATE_LIMIT`	Requests per minute per IP	`100`	`200`

Device Integration Variables (Optional)

Configure devices via UI after deployment, or pre-configure here:

Palo Alto NGFW:

```
PALOALTO_API_URL=https://your-firewall.example.com
PALOALTO_API_KEY=your-api-key
PALOALTO_VERIFY_SSL=true
```

Microsoft Entra ID:

```
ENTRAID_TENANT_ID=your-tenant-id
ENTRAID_CLIENT_ID=your-client-id
ENTRAID_CLIENT_SECRET=your-client-secret
```

SIEM (Splunk/Elastic):

```
SIEM_TYPE=splunk
SIEM_API_URL=https://your-siem.example.com:8089
SIEM_USERNAME=admin
SIEM_PASSWORD=your-password
```

Frontend Service Variables

Variable	Description	Example
`VITE_API_BASE_URL`	Backend API URL	`https://your-backend.railway.app`

🎉 Post-Deployment

1. Verify Deployment

Check Backend Health

```
curl https://your-backend.railway.app/health
# Should return: {"status": "healthy"}
```

Check API Documentation

Visit: `https://your-backend.railway.app/docs`

This opens the interactive Swagger/OpenAPI documentation.

Check Frontend

Visit: `https://your-frontend.railway.app`

You should see the SOaC Framework login page.

2. First Login

1. Open your frontend URL
2. Login with:
   - Username: `admin`
   - Password: `admin123`
3. **Immediately change the password!**
   - Go to **Profile** → **Change Password**

3. Explore Sample Data

The deployment automatically creates:
- ✅ 2 users (admin, analyst)

- ✅ 6 sample devices (Palo Alto, Entra ID, SIEM)
- ✅ 11 sample detection rules
- ✅ 3 sample incidents
- ✅ 1 sample playbook execution

4. Configure Real Devices (Optional)

If you want to connect real security devices:

1. Set `MOCK_MODE=false` in backend environment variables
2. Go to **Devices** page in the UI
3. Add or edit devices with real credentials
4. Test connections
5. Enable background event collection

5. Set Up Monitoring

1. Check Railway logs for errors:
   - Each service → **"Deployments"** → Latest deployment → **"View Logs"**
2. Set up Railway metrics:
   - Each service → **"Metrics"** tab
3. Configure alerts (optional):
   - Project Settings → **"Alerts"**

🔧 Troubleshooting

Issue 1: "Database connection failed"

Symptoms: Backend logs show PostgreSQL connection errors

Solution:
1. Verify PostgreSQL service is running:
- Check database service status in Railway dashboard
2. Verify `DATABASE_URL` is set:
- Backend service → **"Variables"** → Look for `DATABASE_URL`
3. If missing, reconnect database:
- Database service → **"Connect"** → Select backend service

Issue 2: "CORS Error" or "Network Request Failed"

Symptoms: Frontend can't connect to backend, browser console shows CORS errors

Solution:
1. Verify `FRONTEND_URL` in backend variables matches your actual frontend URL
2. Verify `VITE_API_BASE_URL` in frontend variables matches your backend URL
3. Redeploy both services after updating variables

Check URLs:

```
# Backend variables should have:
FRONTEND_URL=https://soac-frontend-production.up.railway.app

# Frontend variables should have:
VITE_API_BASE_URL=https://soac-backend-production.up.railway.app
```

Issue 3: “401 Unauthorized” Errors

Symptoms: Can’t login, or getting 401 errors after login

Solutions:
1. Verify `SECRET_KEY` is set in backend variables
2. SECRET_KEY must be at least 32 characters
3. If you changed SECRET_KEY, clear browser cookies and try again
4. Check backend logs for detailed error messages

Issue 4: Deployment Fails with “Build Error”

Symptoms: Deployment fails during build phase

Solutions:
1. Check deployment logs for specific error
2. Common issues:
- **Dockerfile not found**: Verify Root Directory and Dockerfile Path settings
- **Port binding issues**: Railway auto-assigns ports, no need to configure
- **Memory issues**: Upgrade to a larger Railway plan if needed

Verify Service Settings:
- Backend Root Directory: `backend`
- Backend Dockerfile: `Dockerfile.railway`
- Frontend Root Directory: `frontend`
- Frontend Dockerfile: `Dockerfile.railway`

Issue 5: “Health Check Failed”

Symptoms: Service shows as unhealthy in Railway

Solution:
1. Check if the service is listening on the correct port:
- Railway provides `PORT` environment variable
- Backend should use: `${PORT:-8000}`
2. Verify health check endpoint works:
`bash`
`   curl https://your-backend.railway.app/health`
3. Check service logs for startup errors

Issue 6: Frontend Shows “Cannot Connect to API”

Symptoms: Frontend loads but can’t fetch data

Solutions:
1. Verify backend is deployed and healthy
2. Check `VITE_API_BASE_URL` in frontend variables
3. Test backend directly:
`bash`
`   curl https://your-backend.railway.app/docs`
4. Check browser console for detailed errors
5. Verify CORS configuration (see Issue 2)

Issue 7: Database Initialization Errors

Symptoms: Backend starts but database is empty, no sample data

Solution:
1. Check backend logs for initialization errors
2. Manually trigger database initialization:
- Railway doesn’t have direct shell access, but you can:
- Add a temporary endpoint in your code to trigger `init_db()`
- Or wait for next deployment (init runs on startup)

Issue 8: “Out of Memory” or “Service Crashed”

Symptoms: Service crashes, logs show OOM (Out of Memory) errors

Solutions:
1. Upgrade Railway plan for more memory
2. Optimize application:
- Reduce worker count in `entrypoint.railway.sh`
- Set `DB_POOL_SIZE=5` (instead of 10)
3. Check for memory leaks in logs

General Debugging Tips

View Logs

```
Railway Dashboard → Your Service → Deployments → View Logs
```

Check Environment Variables

```
Railway Dashboard → Your Service → Variables
```

Test Backend API

```
# Health check
curl https://your-backend.railway.app/health

# API docs
open https://your-backend.railway.app/docs

# Test login
curl -X POST https://your-backend.railway.app/api/auth/login \
  -H "Content-Type: application/json" \
  -d '{"username": "admin", "password": "admin123"}'
```

Check Database Connection

Railway provides a database connection string. To verify:
1. Go to PostgreSQL service in Railway
2. Click **"Connect"**
3. Use the connection details to connect with a PostgreSQL client

---

💰 Cost & Resource Management

Free Tier Limits

Railway offers **$5 free credit per month**:
- ~500 hours of runtime for small services
- PostgreSQL database included
- Suitable for development and testing
- **No credit card required initially**

Estimated Usage

For SOaC Framework:
- **Backend Service**: ~$2-3/month
- **Frontend Service**: ~$1-2/month
- **PostgreSQL Database**: ~$1-2/month
- **Total**: ~$4-7/month (within free tier!)

Tips to Stay Within Free Tier

1. **Use Sleep Mode**: Railway sleeps inactive services (saves credits)
2. **Optimize Resources**: Use minimal resources for testing
3. **Monitor Usage**: Check Railway dashboard for credit usage
4. **Scale When Needed**: Upgrade to paid plan when ready for production

Upgrading

When you outgrow the free tier:
1. Go to **Project Settings** → **Billing**
2. Add a payment method
3. Choose a plan:
- **Hobby**: $5/month + usage
- **Pro**: $20/month + usage
4. Get more resources and features

🚀 Advanced Configuration

Custom Domains

1. Go to service **Settings** → **Networking**
2. Click **“Custom Domain”**
3. Add your domain (e.g., `soac.yourdomain.com`)
4. Update DNS records as shown by Railway
5. Railway automatically provisions SSL certificate

Auto-Deploy from GitHub

Railway automatically deploys when you push to your GitHub repository!

Configure Branch:
1. Service **Settings** → **Source**
2. Select **Branch** (default: `main`)
3. Every push to this branch triggers deployment

Environment-Specific Deployments

Create multiple Railway projects for different environments:
- **Development**: `main` branch, `ENVIRONMENT=development`
- **Staging**: `staging` branch, `ENVIRONMENT=staging`
- **Production**: `production` branch, `ENVIRONMENT=production`

Rollback Deployments

1. Go to **Deployments** tab
2. Find the previous successful deployment
3. Click **“Redeploy”**
4. Railway will rollback to that version

📚 Additional Resources

Railway Documentation

- Railway Docs (https://docs.railway.app/)
- Railway CLI (https://docs.railway.app/develop/cli)
- Railway Templates (https://railway.app/templates)

SOaC Framework Documentation

- Main README (./README.md)
- Deployment Guide (./DEPLOYMENT.md) - For other platforms
- Quick Start Guide (./QUICKSTART.md)
- API Documentation (https://your-backend.railway.app/docs) - After deployment

Support

- **Railway Support**: railway.app/help (https://railway.app/help)
- **SOaC Framework Issues**: GitHub Issues

- **Community**: Railway Discord server

✅ Deployment Checklist

Use this checklist to ensure successful deployment:

Pre-Deployment

- [ ] Repository forked/cloned to your GitHub account
- [ ] Railway account created
- [ ] Reviewed `.env.production.example` file

Railway Setup

- [ ] New Railway project created
- [ ] GitHub repository connected
- [ ] PostgreSQL database added
- [ ] Backend service configured (Root Dir: `backend`, Dockerfile: `Dockerfile.railway`)
- [ ] Frontend service configured (Root Dir: `frontend`, Dockerfile: `Dockerfile.railway`)

Environment Variables

- [ ] Backend: `SECRET_KEY` generated and set (32+ characters)
- [ ] Backend: `ALGORITHM=HS256` set
- [ ] Backend: `ACCESS_TOKEN_EXPIRE_MINUTES=1440` set
- [ ] Backend: `ENVIRONMENT=production` set
- [ ] Backend: `FRONTEND_URL` set to frontend Railway URL
- [ ] Backend: `MOCK_MODE=true` set (for testing)
- [ ] Frontend: `VITE_API_BASE_URL` set to backend Railway URL

Deployment

- [ ] Both services deployed successfully
- [ ] No errors in deployment logs
- [ ] Health checks passing
- [ ] Domains generated and accessible

Post-Deployment

- [ ] Backend health endpoint returns 200 OK
- [ ] Frontend loads successfully
- [ ] Login with default credentials works
- [ ] Admin password changed
- [ ] Sample data visible (devices, rules, incidents)
- [ ] API documentation accessible at `/docs`

Optional

- [ ] Custom domain configured (if needed)
- [ ] Real device credentials configured (if not using mock mode)
- [ ] Monitoring and alerts set up
- [ ] Team members invited to Railway project

🎓 Next Steps

After successful deployment:

1. **Explore the Dashboard**
   - Review sample incidents
   - Check detection rules
   - View device connections

2. **Configure Real Devices** (Optional)
   - Add Palo Alto NGFW credentials
   - Connect Entra ID tenant
   - Integrate SIEM

3. **Customize Use Cases**
   - Review operational models
   - Adjust detection rules
   - Create custom playbooks

4. **Set Up Monitoring**
   - Configure alerts
   - Review logs regularly
   - Monitor Railway metrics

5. **Scale as Needed**
   - Monitor credit usage
   - Upgrade plan when ready
   - Optimize resource allocation

🙏 Acknowledgments

Thank you for choosing SOaC Framework! We hope Railway deployment makes your life easier.

Happy Security Operations! 🔒🚀

📝 Deployment Notes

Railway-Specific Configurations

Port Binding

Railway provides a dynamic `PORT` environment variable. Both backend and frontend Dockerfiles are configured to use this:

```
PORT=${PORT:-8000}  # Backend defaults to 8000
PORT=${PORT:-3000}  # Frontend defaults to 3000
```

Database Connection

Railway automatically injects `DATABASE_URL` when PostgreSQL is added. Format:

```
postgresql://user:password@host:port/database
```

Health Checks

Both services include health check endpoints:
- Backend: `/health`
- Frontend: `/` (nginx serves index.html)

Build Process

Railway uses Docker multi-stage builds:
- **Backend**: Python 3.11-slim + Gunicorn + Uvicorn workers
- **Frontend**: Node 18 build → Nginx Alpine serve

Automatic HTTPS

Railway automatically provisions SSL certificates for all generated domains.

Need Help? Open an issue on GitHub or contact the SOaC Framework team!