# iSly Al Platform - Docker Deployment Summary

# **®** What Has Been Created

I've successfully created a complete Docker deployment configuration for your iSly Al agent platform. Here's what's been set up:

### Files Created

### 1. Docker Configuration Files:

- Dockerfile Multi-stage NextJS web application container
- Dockerfile.agents Al agents worker service container
- docker-compose.yml Orchestrates all services
- .dockerignore Optimizes build process
- .env.docker Environment variables template

#### 2. Database Setup:

- docker/db/init.sql Database initialization script
- PostgreSQL with proper schema and sample data

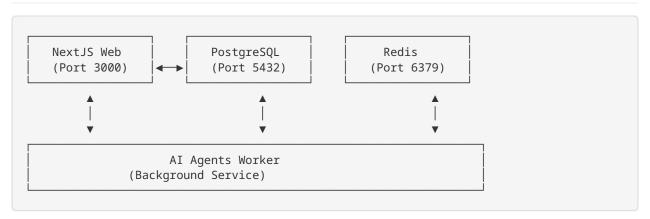
#### 3. Application Files:

- package.json Node.js dependencies
- next.config.js NextJS configuration for Docker
- workers/ai-agents.js Al agents background service
- app/api/health/route.js Health check endpoint

#### 4. Deployment Scripts:

- build\_and\_deploy.sh Automated deployment script
- run\_deployment.sh Script runner
- DOCKER\_DEPLOYMENT\_GUIDE.md Comprehensive documentation

# 髉 Architecture Overview





### **Step 1: Update Environment Variables**

Edit .env.docker and replace placeholder values:

```
OPENAI_API_KEY=your_actual_openai_api_key
ANTHROPIC_API_KEY=your_actual_anthropic_api_key
POSTGRES_PASSWORD=your_secure_password
NEXTAUTH_SECRET=your_secure_random_string
```

### **Step 2: Deploy with Docker Desktop**

```
# Make deployment script executable
chmod +x build_and_deploy.sh
# Run the deployment
./build_and_deploy.sh
```

### **Step 3: Access Your Application**

- Web Interface: http://localhost:3000
- Health Check: http://localhost:3000/api/health



# Key Features

### Multi-Service Architecture

- Web Service: NextJS application with production optimizations
- Database Service: PostgreSQL with automatic initialization
- Cache Service: Redis for session management and job queues
- Worker Service: Al agents processing background tasks

### **Production-Ready Features**

- Multi-stage Docker builds for optimized image sizes
- · Health checks for all services
- · Automatic database initialization
- · Volume persistence for data
- · Network isolation and service discovery
- · Graceful shutdown handling

### **Development-Friendly**

- · Hot-reload support for development
- Comprehensive logging
- Easy service management
- Database migration support
- · Environment variable management

# **■ Service Details**

Service	Container Name	Port	Health Check
Web App	isly-web	3000	/api/health
Database	isly-postgres	5432	pg_isready
Redis	isly-redis	6379	redis-cli ping
Al Agents	isly-ai-agents	8080	/health

# **Management Commands**

### **Basic Operations**

```
# Start all services
docker compose --env-file .env.docker up -d
# Stop all services
docker compose down
# View logs
docker compose logs -f
# Restart specific service
docker compose restart web
```

## **Database Operations**

```
# Access database shell
docker exec -it isly-postgres psql -U isly_user -d isly_db
# Run migrations
docker exec -it isly-web npx prisma migrate deploy
# Backup database
docker exec -it isly-postgres pg_dump -U isly_user isly_db > backup.sql
```

# **Monitoring**

```
# Check service status
docker compose ps
# Monitor resource usage
docker stats
# View specific service logs
docker compose logs web
docker compose logs ai-agents
```



# Verification Checklist

After deployment, verify these items:

- [ ] All containers are running ( docker compose ps )
- [ ] Web application accessible at http://localhost:3000
- [ ] Health check returns 200 OK at http://localhost:3000/api/health
- [ ] Database accepts connections
- [ ] Redis responds to ping
- [ ] Al agents worker is processing jobs
- [ ] No error messages in logs



# Troubleshooting

### Common Issues and Solutions

- 1. Port conflicts: Change ports in docker-compose.yml
- 2. Build failures: Check Dockerfile syntax and dependencies
- 3. Database connection issues: Verify credentials and network connectivity
- 4. Memory issues: Increase Docker Desktop memory allocation
- 5. **Permission errors**: Check file permissions and user contexts

### **Getting Help**

- Check container logs: docker compose logs [service-name]
- Inspect container: docker exec -it [container-name] /bin/sh
- Review Docker Desktop dashboard for visual monitoring



# **E** Success Indicators

Your deployment is successful when:

- Docker Desktop shows all containers as "Running" (green status)
- Web interface loads without errors
- Health check endpoint returns healthy status
- Database queries execute successfully
- Al agents process tasks in the background



### 📚 Next Steps

- 1. Customize Configuration: Adjust settings in docker-compose.yml for your needs
- 2. Add SSL/TLS: Configure reverse proxy for production deployment
- 3. Scale Services: Use Docker Swarm or Kubernetes for production scaling
- 4. Monitor Performance: Set up logging and monitoring solutions
- 5. Backup Strategy: Implement regular database and volume backups

# 🔐 Security Notes

- · Change all default passwords before production use
- Use Docker secrets for sensitive data in production

- Enable firewall rules to restrict access
- Regularly update base images for security patches
- Monitor container logs for suspicious activity

**Deployment Status**: **✓** Ready for Docker Desktop deployment

**Estimated Setup Time**: 5-10 minutes

Resource Requirements: 4GB RAM, 2GB disk space