# AutoProjectManagement

Unified Docker Setup Guide

Comprehensive Guide for Automated and Manual Docker Setup

Automated Setup | Manual Setup | Security | Monitoring

# Quick Navigation

Explore the comprehensive Docker setup guide sections

### Automated Setup
Zero-configuration deployment

### Manual Setup
Step-by-step instructions

### Architecture
System design and components

### Configuration
Environment and customization

### Monitoring
Health checks and maintenance

### Troubleshooting
Common issues and solutions

# Automated Setup - One-Command Deployment

Zero-configuration deployment for all platforms

## Platform Commands

### Linux/macOS

```
./scripts/one-command-deploy.sh
```

### Windows Command Prompt

```
scripts\auto-docker-setup.bat
```

### Windows PowerShell

```
.\scripts\auto-docker-setup.ps1
```

## Zero-Configuration Features

- Auto-detects environment
- Creates .env file
- Generates SSL certificates
- Sets up monitoring
- Configures logging
- Handles port conflicts
- Installs Docker if needed
- Backup/restore functionality

# Automated Setup - Environment Auto-Detection

System automatically detects your environment and applies appropriate configuration

The system **automatically detects** your environment based on git branch or manual flags, ensuring the right configuration is applied without manual intervention.

| Environment | Detection Method | Configuration Used |
|---|---|---|
| Development | Git branch != main/master | `docker-compose.dev.yml` |
| Production | Git branch = main/master | `docker-compose.prod.yml` |
| Override | Use --dev or --prod flags | `Manual selection` |

Override flags take precedence over git branch detection, allowing manual control when needed

# Automated Setup - Available Commands

Quick shortcuts for development and production environments

## Development Commands

### Start Development

```
./scripts/one-command-deploy.sh --dev
```

Initializes development environment with all services

### View Logs

```
./scripts/apm-dev.sh logs api
```

Displays real-time logs for API service

### Access Container

```
./scripts/apm-dev.sh shell
```

Opens interactive shell in API container

### Show Status

```
./scripts/apm-dev.sh status
```

Displays health status of all services

### Stop Services

```
./scripts/apm-dev.sh stop
```

Stops all development services

## Production Commands

### Start Production

```
./scripts/one-command-deploy.sh --prod
```

Initializes production environment with SSL enabled

### View Logs

```
./scripts/apm-prod.sh logs api
```

Displays real-time logs for API service

### Update Services

```
./scripts/apm-prod.sh update
```

Pulls latest images and updates services

### Create Backup

```
./scripts/apm-prod.sh backup
```

Creates backup of data and configuration

### Stop Services

```
./scripts/apm-prod.sh stop
```

Stops all production services

# Manual Setup - Prerequisites

System requirements and installation commands

Before proceeding with manual setup, ensure your system meets the **minimum requirements** for optimal performance.

| Component | Minimum | Recommended |
|---|---|---|
| Docker Engine | 20.10+ | 24.0+ |
| Docker Compose | 2.0+ | 2.20+ |
| CPU Cores | 2 | 4+ |
| RAM | 4GB | 8GB+ |
| Storage | 20GB | 50GB+ |
| OS | Linux/macOS/Windows | Linux (Ubuntu 22.04+) |

### Installation Commands

```
# Check Docker installation
docker --version
docker compose version
# Install Docker (Ubuntu/Debian)
curl -fsSL https://get.docker.com -o get-docker.sh
sudo sh get-docker.sh
sudo usermod -aG docker $USER
# Install Docker Compose
sudo apt-get update
sudo apt-get install docker-compose-plugin
```

# Manual Setup - Directory Structure

Organized project layout for Docker deployment

The **AutoProjectManagement** project follows a structured directory layout to organize Docker configurations, scripts, and application files.

```
AutoProjectManagement/
  docker/
    api/
      Dockerfile
      entrypoint.sh
      healthcheck.sh
    worker/
      Dockerfile
      entrypoint.sh
    monitor/
      Dockerfile
      entrypoint.sh
    nginx/
      Dockerfile
      nginx.conf
      ssl/
  scripts/
    auto-docker-setup.sh
    one-command-deploy.sh
    apm-dev.sh
    apm-prod.sh
    backup.sh
    auto-docker-setup.bat
    auto-docker-setup.ps1
  docker-compose.yml
  docker-compose.dev.yml
  docker-compose.prod.yml
  .dockerignore
  .env.example
  .gitignore
```

The project structure separates configuration files, scripts, and Docker components for better organization and maintainability

# Manual Setup - Deployment Steps

Step-by-step instructions for manual Docker deployment

## 1 Clone Repository

```
git clone <repository-url>
cd AutoProjectManagement
```

## 2 Create Environment File

```
cp .env.example .env
# Edit .env with your configuration
nano .env
```

## 3 Development Deployment

### Start Services

```
docker-compose -f docker-compose.dev.yml up --build
```

### View Logs

```
docker-compose logs -f api
```

### Scale Workers

```
docker-compose up -d --scale worker=3
```

## 4 Production Deployment

### Deploy Services

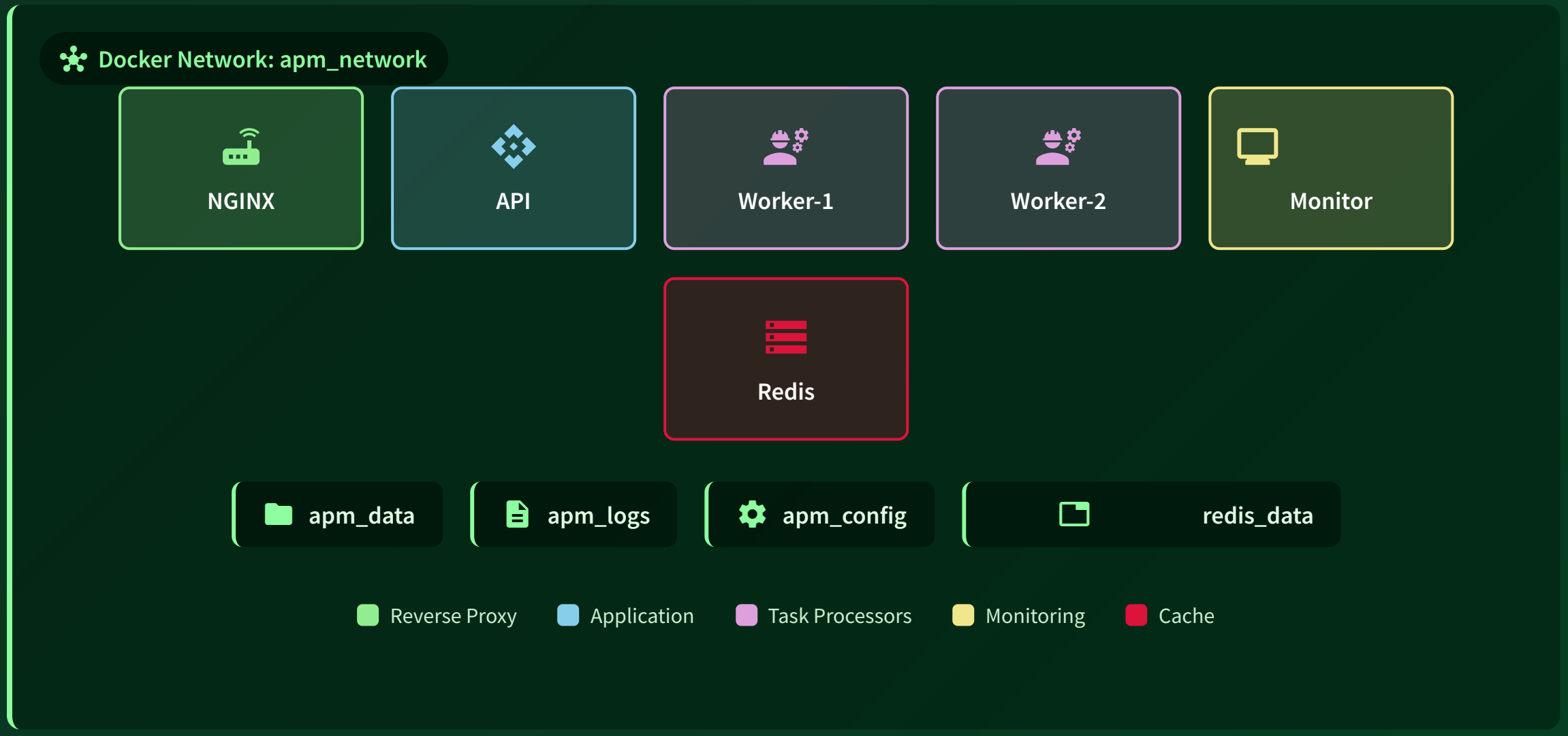
```
docker-compose -f docker-compose.prod.yml pull
docker-compose -f docker-compose.prod.yml up -d --build
```

### Health Check

```
./scripts/health_check.sh
```

# Architecture - High-Level Architecture

Docker network structure with interconnected services and volumes



# Architecture - Service Architecture Detail

Internal components and connections of each service

## API Service

**FastAPI Application**
Core REST API framework

**Configuration Loader**
Manages environment settings

**Service Modules**
Business logic implementation

**JSON Database**
Data persistence layer

## Worker Service

**Task Processor**
Handles background jobs

**Job Queue**
Manages task prioritization

**Execution Engine**
Runs task operations

**Notification Service**
Sends status updates

## Monitor Service

**Health Checker**
Monitors service status

**Metrics Collector**
Gathers performance data

**Alert Manager**
Handles issue notifications

Services communicate through **well-defined interfaces** with proper separation of concerns for maintainability and scalability

# Configuration - Environment Variables

Key environment variables organized by purpose

## Application

**ENVIRONMENT** `production`

**DEBUG** `false`

**LOG_LEVEL** `INFO`

## API

**API_HOST** `0.0.0.0`

**API_PORT** `8000`

**API_WORKERS** `4`

## Database

**DATA_PATH** `/app/data`

**BACKUP_INTERVAL** `3600`

**MAX_BACKUPS** `10`

## Redis

**REDIS_URL** `redis://redis:6379`

**REDIS_DB** `0`

## Monitoring

**MONITOR_INTERVAL** `30`

**ALERT_WEBHOOK_URL** `https://hooks.slack.com/...`

## Security

**SECRET_KEY** `your-secret-key-here`

**JWT_SECRET** `your-jwt-secret-here`

**ALLOWED_HOSTS** `localhost,127.0.0.1`

## SSL Configuration (Production)

**SSL_CERT_PATH** `/etc/nginx/ssl/cert.pem`

**SSL_KEY_PATH** `/etc/nginx/ssl/key.pem`

Environment variables provide **flexible configuration** without modifying code. Copy **.env.example** to **.env** and customize as needed.

# Configuration - Docker Compose Configuration

Key components of docker-compose.yml file

## Services

### nginx
- Ports: 80, 443
- SSL & logs volumes
- Depends on: api

### api
- Port: 8000
- Data, logs, config volumes
- Health check enabled

### worker-1
- Data & logs volumes
- Docker socket access
- Depends on: api

### worker-2
- Data & logs volumes
- Docker socket access
- Depends on: api

### monitor
- Logs volume
- Docker socket access
- Depends on: api, workers

### redis
- Port: 6379
- Data volume
- Append-only mode

## Volumes

**apm_data**
Application data

**apm_logs**
Log files

**apm_config**
Configuration files

**redis_data**
Redis storage

## Network

**apm_network**
Bridge network

**Subnet**
172.20.0.0/16

Docker Compose creates an **isolated environment** with dedicated network and persistent volumes for data storage and configuration

# Monitoring & Maintenance - Health Check Endpoints

Service health monitoring endpoints and expected responses

Each service provides **dedicated health endpoints** for monitoring system status and performance. These endpoints are used by the monitoring service and external tools.

| Service | Endpoint | Method | Expected Response | Timeout |
|---|---|---|---|---|
| API | /health | GET | {"status": "healthy"} | 5s |
| Worker-1 | /health | GET | {"status": "healthy"} | 5s |
| Worker-2 | /health | GET | {"status": "healthy"} | 5s |
| Monitor | /metrics | GET | Prometheus metrics | 10s |

Health checks are automatically performed by the monitoring service at regular intervals. Failed health checks trigger alerts and notifications.

# Monitoring & Maintenance - Access URLs

Service access URLs for development and production environments

Different services are accessible through specific URLs in **development** and **production** environments. Production uses HTTPS with proper SSL certificates.

| Service | Development | Production |
|---|---|---|
| API | `http://localhost:8000` DEV | `https://localhost/api` PROD |
| Monitor | `http://localhost:8080` DEV | `https://localhost/monitor` PROD |
| Web Interface | `http://localhost:8080` DEV | `https://localhost` PROD |

Production URLs are automatically configured with SSL certificates during setup. Development URLs use HTTP for easier local testing.

# Monitoring & Maintenance - Maintenance Schedule

Regular maintenance tasks to ensure system health and performance

Regular **maintenance tasks** help ensure optimal performance, security, and reliability of the Docker environment.

| Task | Frequency | Command | Duration |
|---|---|---|---|
| Log rotation | Daily | `docker exec api logrotate` | 1 min |
| Backup creation | Daily | `./scripts/backup.sh` | 5-10 min |
| Image updates | Weekly | `docker-compose pull` | 2-5 min |
| Volume cleanup | Monthly | `docker volume prune` | 1-2 min |
| Security scan | Monthly | `docker scan` | 5-10 min |

Most maintenance tasks can be automated through cron jobs or similar scheduling tools to ensure consistent execution without manual intervention.

# Troubleshooting - Common Issues and Solutions

Identify and resolve frequent Docker deployment problems

When encountering issues with your Docker setup, refer to this guide for **quick solutions** and **preventive measures**.

| Issue | Symptoms | Solution | Prevention |
|---|---|---|---|
| **Container won't start** | Exit code 1 | Check logs: `docker logs container_name` | Validate configuration |
| **Port conflicts** | "Port already in use" | Change port mapping | Use dynamic ports |
| **Volume permission denied** | Permission errors | Fix volume permissions | Use proper user IDs |
| **Out of disk space** | "No space left on device" | Clean up images/volumes | Set up monitoring |
| **Network issues** | "Connection refused" | Check network configuration | Use docker networks |

For additional debugging commands, refer to the **Debug Commands** and **Log Analysis** sections in the documentation.

# Security Considerations - Security Architecture

Multi-layered security approach for Docker deployment

The Docker setup implements a **defense-in-depth** strategy with multiple security layers to protect against various threats.

## Network Security

- Firewall Rules
- VPN Access
- TLS Encryption

## Container Security

- Non-root User
- Minimal Base Image
- Vulnerability Scanning

## Application Security

- Authentication
- Authorization
- Input Validation

## Data Security

- Data Encryption
- Secure Backups
- Access Control

## Security Checklist

- Use non-root user in containers
- Scan images for vulnerabilities
- Enable TLS/SSL
- Keep base images updated
- Use secrets management
- Implement proper authentication

# Support & Resources

Quick help commands and additional resources for troubleshooting

## Quick Help Commands

```
./scripts/apm-dev.sh status
```

```
./scripts/apm-dev.sh logs
```

```
./scripts/apm-dev.sh info
```

```
./scripts/apm-prod.sh backup
```

## Additional Resources

- Troubleshooting Guide
- Security Documentation
- Performance Tuning
- Architecture Details

## Getting Help

1. **Check this guide**
   Find your specific issue
2. **Run diagnostic commands**
   Gather system information
3. **Check logs**
   Look for error messages
4. **Create an issue**
   Include system info and errors

For the most up-to-date information and community support, visit the project repository and check the **issues** and **discussions** sections.