OpenFarm - First Time Setup Guide

This guide will walk you through setting up the OpenFarm project for the first time.

Table of Contents

Prerequisites

Before you begin, ensure you have the following installed on your system:

Required Software

1. **Docker** (version 20.10 or later)
   - Install from Docker Desktop or your distribution's package manager
   - Verify installation: `docker --version` and `docker compose version`
2. **Git** (for cloning the repository)
   - Verify installation: `git --version`

Optional Software

> **Note**: The following are only needed if you plan to do local development outside of Docker containers.

- **.NET SDK 9.0** (for local development of C# services)
  - Download from Microsoft .NET Downloads
  - Verify installation: `dotnet --version` (should show 9.0.x)
  - Required for: Building/running C# services locally, developing the native desktop app
- **Python 3.11+** (for local development of GcodeOracle service)
  - Download from Python Downloads
  - Verify installation: `python3 --version`
  - Required for: Running GcodeOracle service locally outside Docker
- **IDE/Editor**: Visual Studio, Rider, or VS Code with C# extensions
- **PostgreSQL Client**: For direct database access (pgAdmin, DBeaver, or psql CLI)

Initial Setup

1. Clone the Repository

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

2. Checkout the Development Branch

```
git checkout dev
```

3. Create Environment File

Copy the environment template to create your `.env` file:

```
cp env.template .env
```

Important: The `.env` file contains sensitive information and should never be committed to version control. It is already in `.gitignore`.

Environment Configuration

Edit the `.env` file and configure all the required values. The file contains detailed comments explaining each setting.

Essential Configuration Steps

1. **PostgreSQL Database Settings**
   - Set `POSTGRES_DB`, `POSTGRES_USER`, and `POSTGRES_PASSWORD`
   - Use strong, unique passwords
   - The `DATABASE_CONNECTION_STRING` will be automatically constructed from these values
2. **RabbitMQ Settings**
   - Set `RABBITMQ_USER` and `RABBITMQ_PASSWORD`
   - Use strong, unique passwords
   - `RABBITMQ_HOST` should remain `rabbitmq` (the Docker service name)
3. **MinIO Object Storage**
   - Set `MINIO_ROOT_USER` and `MINIO_ROOT_PASSWORD` (minimum 8 characters)
   - `MINIO_USE_SSL` should be `false` for local development
4. **File Server Configuration**
   - `FILE_SERVER_BASE_URL` should remain `http://file-processor:80` for Docker setup
   - Adjust `MAX_FILE_SIZE_MB`, `DOWNLOAD_TIMEOUT_SECONDS`, and `MAX_CONCURRENT_DOWNLOADS` as needed
   - Configure `ALLOWEDORIGINS__0`, `ALLOWEDORIGINS__1`, `ALLOWEDORIGINS__2` for your frontend URLs (the defaults should work fine)
5. **Email Service** (if using email functionality)
   - Set `GMAIL_EMAIL` to your Gmail address
   - Generate a Gmail App Password at Google App Passwords
   - Set `GMAIL_APP_PASSWORD` to the generated app password
   - Configure `SENDER_NAME`, `SMTP_SERVER`, `SMTP_PORT`, `IMAP_SERVER`, `IMAP_PORT`
   - Set `COMPANY_LOGO_URL` to your company logo URL
6. **Discord Service** (if using Discord notifications)
   - Create a Discord bot at Discord Developer Portal
   - Set `DISCORD_TOKEN` to your bot token
   - Set `DISCORD_CHANNEL` to your Discord channel ID
7. **Google Form Listener**
   - Set `GOOGLE_FORM_ID` (extract from your Google Form URL)
   - See Google Form Listener README for detailed form setup instructions
8. **Payment Processing Service** (if using payment functionality)
   - Set `PAYMENT_API_KEY` to your payment processing API key, a minimum 32 character string
   - Set `ASPNETCORE_CERT_PASSWORD` for HTTPS certificate (if using HTTPS)
   - Ensure the certificate file is available in the service

Google Service Account Setup

The Google Service Account is required for Google Form integration and Google Drive file access.

Steps

1. **Create a Google Cloud Project**
   - Go to Google Cloud Console
   - Create a new project (e.g., "OpenFarm - Your Institution")
2. **Create a Service Account**
   - Navigate to "IAM & Admin" → "Service Accounts"
   - Click "+ Create Service Account"
   - Name it (e.g., "OpenFarm Service Account")
   - Grant it the "Owner" role
3. **Generate Service Account Key**
   - Select the created service account
   - Go to the "Keys" tab
   - Click "Add Key" → "Create new key"
   - Choose JSON format
   - Download the JSON file
4. **Place the Credentials File**
   - Copy the downloaded JSON file to:
     - `google-form-listener/` directory
     - `FileServer/FileProcessor/` directory
   - Rename it to match `GOOGLE_SERVICE_ACCOUNT_CRED_FILE` in your `.env` file (default: `google-service-account-credentials.json`)
5. Share Google Form with Service Account

5. Share Google Form with Service Account
    - Open your Google Form in edit mode
    - Click the share button (top right)
    - Add the service account email (found in the JSON file under `client_email`) as an editor
    - See Google Form Listener README for complete form setup instructions

Running the Services

Start All Services with Docker Compose

The easiest way to run all services is using Docker Compose:

```
docker compose up -d
```

This will:

- Build all Docker images
- Start all services in the correct order (respecting dependencies)
- Create necessary volumes and networks
- Initialize the database

View Service Logs

To view logs from all services:

```
docker compose logs -f
```

To view logs from a specific service:

```
docker compose logs -f <service-name>
```

For example:

```
docker compose logs -f file-processor
docker compose logs -f email-service
```

Stop All Services

```
docker compose down
```

To also remove volumes (Warning: This will delete all data):

```
docker compose down -v
```

Service Health Checks

Most services include health checks. You can verify service status:

```
docker compose ps
```

Verifying the Setup

1. Check Service Status

```
docker compose ps
```

All services should show as "healthy" or "running".

2. Access Service UIs

- **RabbitMQ Management**: http://localhost:15672
    - Login with `RABBITMQ_USER` and `RABBITMQ_PASSWORD` from your `.env`
- **MinIO Console**: http://localhost:9001
    - Login with `MINIO_ROOT_USER` and `MINIO_ROOT_PASSWORD` from your `.env`
- **File Processor API**: http://localhost:5001

  - Health check: http://localhost:5001/health

3. Verify Database Connection

Connect to PostgreSQL using the credentials from your `.env`:

```
docker exec -it postgres-container psql -U <POSTGRES_USER> -d <POSTGRES_DB>
```

Or use a PostgreSQL client with:

- Host: `localhost`
- Port: `5432` (or your `POSTGRES_PORT`)
- Database: `POSTGRES_DB` value
- Username: `POSTGRES_USER` value
- Password: `POSTGRES_PASSWORD` value

Native Desktop App

The native desktop app (Avalonia-based) can be run locally for development:

```
cd native-desktop-app
dotnet run
```

Ensure the app's configuration points to the correct service endpoints.

Troubleshooting

Services Won't Start

1. **Check Docker is running**: `docker ps`
2. **Check port conflicts**: Ensure ports 5432, 5672, 15672, 9000, 9001, 5001 are not in use
3. **Check environment variables**: Verify all required variables are set in `.env`
4. **Check logs**: `docker compose logs <service-name>`

Database Connection Issues

1. **Wait for database to be ready**: The database takes time to initialize
2. **Check health status**: `docker compose ps postgres`
3. **Verify connection string**: Ensure `DATABASE_CONNECTION_STRING` matches your PostgreSQL settings
4. **Check network**: Ensure services are on the same Docker network (`internal-network`)

RabbitMQ Connection Issues

1. **Wait for RabbitMQ to start**: It may take 30-60 seconds
2. **Check credentials**: Verify `RABBITMQ_USER` and `RABBITMQ_PASSWORD` in `.env`
3. **Access management UI**: http://localhost:15672 to verify connectivity

MinIO Issues

1. **Check bucket creation**: The `minio-init` service should create buckets automatically
2. **Verify credentials**: Check `MINIO_ROOT_USER` and `MINIO_ROOT_PASSWORD`
3. **Check console**: Access http://localhost:9001 to verify setup

Google Service Account Issues

1. **Verify file location**: Ensure the JSON file is in both required directories
2. **Check file name**: Must match `GOOGLE_SERVICE_ACCOUNT_CRED_FILE` in `.env`
3. **Verify permissions**: Service account must have access to the Google Form
4. **Check service account email**: Ensure it's added as an editor to the form

Build Failures

1. **Clear Docker cache**: `docker compose build --no-cache`
2. **Check Docker resources**: Ensure Docker has enough memory and CPU allocated
3. **Check .NET SDK version** (if building locally): Must be 9.0 (`dotnet --version`) - Note: Not required if using Docker

Project Structure

- **DatabaseAccess/**: Shared database access library
- **RabbitMQHelper/**: Shared RabbitMQ messaging library

- **EmailService/**: Email sending and receiving service
- **DiscordService/**: Discord bot notifications for farm operators
- **FileServer/**: File upload/download processing service, utilizes MinIO
- **PrinterManagementService/**: Printer and print job management
- **print-submission-processing-service/**: Processes print submissions
- **google-form-listener/**: Monitors Google Forms for new submissions
- **PaymentProcessingService/**: Provides API access to payment information for outside payment handling
- **GcodeOracle/**: G-code rendering service (Python)
- **native-desktop-app/**: Desktop application (Avalonia)

Next Steps

After successful setup:

1. Review individual service READMEs for service-specific documentation:
   - DatabaseAccess README
   - FileServer README
   - PrinterManagementService README
   - Google Form Listener README
   - And others in their respective directories
2. Configure your Google Form according to the Google Form Listener README
3. Set up your printers and configure the printer management system
4. Test the complete workflow from form submission to print job processing

Support

For issues or questions:

- Check individual service READMEs for detailed documentation
- Review Docker logs for error messages
- Verify all environment variables are correctly set
- Ensure all prerequisites are installed and up to date