📋 Manual de Instalación - Coffee Sales Project

Este manual te guiará paso a paso para instalar y ejecutar las aplicaciones **coffee-model**, **coffee-api** y **coffee-dash**.

📋 Prerrequisitos

Software necesario:

- Docker >= 20.10
- Docker Compose >= 2.0
- Git
- Python 3.12+ (para desarrollo local)
- Tox >= 4.0 (para gestión de entornos y testing)
- AWS CLI (para despliegue en AWS)

Verificar instalación:

```
docker --version
docker-compose --version
git --version
python --version
tox --version
aws --version
```

Instalar Tox (si no está instalado):

```
pip install tox
```

🚀 Instalación Rápida con Docker

1. Clonar el repositorio

```
git clone https://github.com/tu-usuario/coffee-sales-project.git
cd coffee-sales-project
```

2. Construir las imágenes

```
# Construir coffee-api
docker build -f apps/coffee-api/Dockerfile -t coffee-api:latest .
```

```
# Construir coffee-dash
docker build -f apps/coffee-dash/Dockerfile -t coffee-dash:latest .
```

3. Crear red Docker

```
docker network create coffee-network
```

4. Ejecutar los servicios

```
# Ejecutar coffee-api
docker run --name coffee-api \
  --network coffee-network \
  -p 8001:8001 \
  -d coffee-api:latest

# Ejecutar coffee-dashboard
docker run --name coffee-dashboard \
  --network coffee-network \
  -p 5006:5006 \
  -d coffee-dash:latest
```

5. Verificar instalación

```
# Ver contenedores ejecutándose
docker ps

# Probar coffee-api
curl http://localhost:8001/api/v1/health

# Acceder al dashboard
# Abrir navegador: http://localhost:5006/dashboard
```

📦 Instalación Individual por Componente

🤖 Coffee-Model (Paquete ML)

Instalación desde código:

```
cd apps/coffee-model

# Instalar dependencias
pip install -r requirements/requirements.txt
```

```
# Construir el paquete
python -m build

# Instalar el paquete
pip install dist/model_coffee_sales_prediction-0.0.3-py3-none-any.whl
```

Con Tox (Recomendado):

```
cd apps/coffee-model

# Entrenar el modelo
tox run -e train

# Ejecutar tests
tox run -e test_package

# Verificar calidad de código
tox run -e checks

# Ejecutar tipo de verificación
tox run -e typechecks

# Ejecutar todos los entornos
tox run
```

Uso del modelo:

```
from model.predict import make_prediction
import pandas as pd

# Crear datos exógenos
exog_data = pd.DataFrame({
    'is_holiday': [0, 0, 0],
    'is_holiday_prev': [0, 0, 0],
    'is_holiday_next': [0, 0, 1],
    'wx_temperature_2m': [22, 24, 23],
    'wx_precipitation': [0, 2, 0],
    'wx_cloudcover': [30, 60, 40]
})

# Hacer predicción
result = make_prediction(horizon=3, exog_data=exog_data)
print(result)
```

⚡ Coffee-API (FastAPI Service)

Instalación local:

```
cd apps/coffee-api

# Instalar dependencias
pip install -r requirements.txt
pip install -r test_requirements.txt

# Instalar el modelo
pip install model-pkg/model_coffee_sales_prediction-0.0.3-py3-none-any.whl

# Ejecutar API
uvicorn app.main:app --host 0.0.0.0 --port 8001 --reload
```

Con Tox:

```
cd apps/coffee-api

# Ejecutar tests
tox run -e test_app

# Verificar calidad de código
tox run -e checks

# Ejecutar tipo de verificación
tox run -e typechecks

# Ejecutar todos los entornos
tox run
```

Con Docker:

```
# Construir imagen
docker build -f apps/coffee-api/Dockerfile -t coffee-api:latest .

# Ejecutar contenedor
docker run --name coffee-api -p 8001:8001 -d coffee-api:latest

# Ver logs
docker logs coffee-api
```

Endpoints disponibles:

- **Health Check**: `GET http://localhost:8001/api/v1/health`
- **Predicciones**: `POST http://localhost:8001/api/v1/predict`
- **Documentación**: `http://localhost:8001/docs`

Ejemplo de uso:

```
curl -X POST http://localhost:8001/api/v1/predict \
  -H "Content-Type: application/json" \
  -d '{
    "horizon": 7,
    "exog_data": [
      {
        "is_holiday": 0,
        "is_holiday_prev": 0,
        "is_holiday_next": 0,
        "wx_temperature_2m": 22.5,
        "wx_precipitation": 0.0,
        "wx_cloudcover": 30.0
      }
    ]
  }'
```

📊 Coffee-Dashboard (Panel App)

Instalación local:

```
cd apps/coffee-dash

# Instalar dependencias
pip install -r requirements.txt

# Configurar URL de API (opcional)
export COFFEE_API_URL=http://localhost:8001

# Ejecutar dashboard
panel serve dashboard.py --port 5006 --allow-websocket-origin=* --show
```

Con Docker:

```
# Construir imagen
docker build -f apps/coffee-dash/Dockerfile -t coffee-dash:latest .

# Ejecutar con API local
docker run --name coffee-dashboard \
  -p 5006:5006 \
  -e COFFEE_API_URL=http://host.docker.internal:8001 \
  -d coffee-dash:latest

# Ejecutar con API externa
docker run --name coffee-dashboard \
  -p 5006:5006 \
```

```
  -e COFFEE_API_URL=http://tu-api-externa.com:8001 \
  -d coffee-dash:latest
```

Acceso:

- **Dashboard**: http://localhost:5006/dashboard

🐳 Instalación con Docker Compose

1. Usar docker-compose.yml

```
# Construir y ejecutar todos los servicios
docker-compose up --build -d

# Ver logs
docker-compose logs -f

# Parar servicios
docker-compose down
```

2. Configuración docker-compose.yml:

```
version: '3.8'

services:
  coffee-api:
    build:
      context: .
      dockerfile: apps/coffee-api/Dockerfile
    ports:
      - "8001:8001"
    networks:
      - coffee-network
    healthcheck:
      test: ["CMD", "curl", "-f", "http://localhost:8001/api/v1/health"]
      interval: 30s
      timeout: 10s
      retries: 3

  coffee-dashboard:
    build:
      context: .
      dockerfile: apps/coffee-dash/Dockerfile
    ports:
      - "5006:5006"
    environment:
      - COFFEE_API_URL=http://coffee-api:8001
```

```
    networks:
      - coffee-network
    depends_on:
      coffee-api:
        condition: service_healthy

networks:
  coffee-network:
    driver: bridge
```

☁️ Despliegue en AWS

1. Configurar AWS CLI

```
aws configure
# Ingresar: Access Key, Secret Key, Region, Output format
```

2. Crear repositorios ECR

```
# Crear repositorio para coffee-api
aws ecr create-repository --repository-name coffee-api --region us-east-1

# Crear repositorio para coffee-dashboard
aws ecr create-repository --repository-name coffee-dashboard --region us-
east-1
```

3. Subir imágenes a ECR

```
# Autenticarse en ECR
aws ecr get-login-password --region us-east-1 | docker login --username
AWS --password-stdin 123456789012.dkr.ecr.us-east-1.amazonaws.com

# Tagear y subir coffee-api
docker tag coffee-api:latest 123456789012.dkr.ecr.us-east-
1.amazonaws.com/coffee-api:latest
docker push 123456789012.dkr.ecr.us-east-1.amazonaws.com/coffee-api:latest

# Tagear y subir coffee-dashboard
docker tag coffee-dash:latest 123456789012.dkr.ecr.us-east-
1.amazonaws.com/coffee-dashboard:latest
docker push 123456789012.dkr.ecr.us-east-1.amazonaws.com/coffee-
dashboard:latest
```

4. Desplegar en ECS

```
# Crear cluster ECS
aws ecs create-cluster --cluster-name coffee-cluster

# Registrar task definitions
aws ecs register-task-definition --cli-input-json file://coffee-api-task-definition.json
aws ecs register-task-definition --cli-input-json file://coffee-dashboard-task-definition.json

# Crear servicios
aws ecs create-service --cluster coffee-cluster --service-name coffee-api --task-definition coffee-api:1 --desired-count 1
aws ecs create-service --cluster coffee-cluster --service-name coffee-dashboard --task-definition coffee-dashboard:1 --desired-count 1
```

🧪 Desarrollo y Testing con Tox

Estructura de entornos Tox disponibles:

Para Coffee-Model (`apps/coffee-model/tox.ini`):

```
# Ver entornos disponibles
tox list

# Entrenar modelo
tox run -e train

# Ejecutar tests del paquete
tox run -e test_package

# Verificar calidad de código (flake8, isort, black, mypy)
tox run -e checks

# Verificar tipos con mypy
tox run -e typechecks

# Limpiar entornos
tox run -e clean
```

Para Coffee-API (`apps/coffee-api/tox.ini`):

```
# Ver entornos disponibles
tox list

# Ejecutar tests de la API
tox run -e test_app
```

```
# Verificar calidad de código
tox run -e checks

# Verificar tipos
tox run -e typechecks

# Limpiar entornos
tox run -e clean
```

Configuración de desarrollo:

1. Configurar entorno de desarrollo:

```
# Crear entorno virtual
python -m venv .venv
source .venv/bin/activate  # Linux/Mac
# .venv\Scripts\activate  # Windows

# Instalar tox
pip install tox

# Instalar dependencias de desarrollo
pip install -r requirements/dev_requirements.txt
```

2. Workflow de desarrollo típico:

```
# 1. Hacer cambios en el código

# 2. Verificar calidad de código
tox run -e checks

# 3. Ejecutar tests
tox run -e test_package  # Para coffee-model
tox run -e test_app      # Para coffee-api

# 4. Verificar tipos
tox run -e typechecks

# 5. Si todo está bien, entrenar modelo (si aplica)
tox run -e train
```

3. Comandos útiles de Tox:

```
# Ejecutar solo un entorno específico
tox run -e checks
```

```
# Ejecutar múltiples entornos
tox run -e checks,test_package

# Forzar recreación de entornos
tox run --recreate

# Ver información detallada
tox run -v

# Ejecutar con variables de entorno
MLFLOW_TRACKING_URI=http://localhost:5000 tox run -e train

# Mostrar configuración de tox
tox config

# Listar entornos disponibles
tox list

# Ejecutar en paralelo (si está configurado)
tox run-parallel
```

Configuraciones de Tox por proyecto:

Coffee-Model tox.ini:

```
[tox]
envlist = checks, typechecks, test_package, train
min_version = 4.0

[testenv]
install_command = pip install {opts} {packages}

[testenv:test_package]
deps =
    -r{toxinidir}/requirements/test_requirements.txt
commands = python -m pytest {posargs}

[testenv:train]
deps =
    -r{toxinidir}/requirements/requirements.txt
commands = python -m model.train_pipeline

[testenv:checks]
deps =
    -r{toxinidir}/requirements/test_requirements.txt
commands =
    flake8 model tests
    isort --check-only --diff model tests
    black --check model tests
    {[testenv:typechecks]commands}
```

```
[testenv:typechecks]
deps =
    -r{toxinidir}/requirements/test_requirements.txt
commands = mypy model
```

Coffee-API tox.ini:

```
[tox]
envlist = checks, typechecks, test_app
min_version = 4.0

[testenv]
install_command = pip install {opts} {packages}

[testenv:test_app]
deps =
    -r{toxinidir}/requirements.txt
    -r{toxinidir}/test_requirements.txt
commands = python -m pytest app/tests {posargs}

[testenv:checks]
deps =
    -r{toxinidir}/requirements.txt
    -r{toxinidir}/test_requirements.txt
commands =
    flake8 app
    isort --check-only --diff app
    black --check app
    {[testenv:typechecks]commands}

[testenv:typechecks]
deps =
    -r{toxinidir}/requirements.txt
    -r{toxinidir}/test_requirements.txt
commands = mypy app
```

🔧 Solución de Problemas

Problemas comunes:

1. Error de arquitectura en EC2

```
# Si obtienes "exec format error"
docker build --platform linux/amd64 -f apps/coffee-api/Dockerfile -t
coffee-api:latest .
```

2. Falta de espacio en disco

```
# Limpiar Docker
docker system prune -a -f
docker volume prune -f

# Ampliar volumen EBS en AWS Console
# Luego en EC2:
sudo growpart /dev/xvda1 1
sudo resize2fs /dev/xvda1
```

3. API no conecta con Dashboard

```
# Verificar red Docker
docker network ls
docker network inspect coffee-network

# Verificar variables de entorno
docker exec coffee-dashboard env | grep COFFEE_API_URL
```

4. Puertos bloqueados en EC2

```
# Verificar Security Groups en AWS Console
# Agregar reglas para puertos 8001 y 5006
```

5. Errores de Tox

```
# Error: "tox: command not found"
pip install tox

# Error: "No tox.ini found"
cd apps/coffee-model  # o apps/coffee-api

# Error: entorno corrupto
tox run --recreate

# Error: dependencias faltantes
pip install -r requirements/requirements.txt
```

6. Errores de importación en tests

```
```

```
# Error: "ModuleNotFoundError"
# Verificar que el paquete esté instalado
pip install -e .

# O usar tox que maneja esto automáticamente
tox run -e test_package
```

📊 Verificación de Instalación

Checklist completo:

- [ ] **Coffee-API ejecutándose**: `curl http://localhost:8001/api/v1/health`
- [ ] **Coffee-Dashboard accesible**: `http://localhost:5006/dashboard`
- [ ] **Predicciones funcionando**: Usar curl de ejemplo
- [ ] **Dashboard conecta con API**: Ver logs sin errores
- [ ] **Contenedores saludables**: `docker ps` muestra "healthy"
- [ ] **Tests pasan**: `tox run -e test_package` y `tox run -e test_app`
- [ ] **Calidad de código**: `tox run -e checks` sin errores
- [ ] **Tipos correctos**: `tox run -e typechecks` sin errores

URLs de acceso:

- **API Health**: http://localhost:8001/api/v1/health
- **API Docs**: http://localhost:8001/docs
- **Dashboard**: http://localhost:5006/dashboard

🎯 Comandos de Gestión

Comandos útiles:

Docker:

```
# Ver logs en tiempo real
docker logs -f coffee-api
docker logs -f coffee-dashboard

# Reiniciar servicios
docker restart coffee-api coffee-dashboard

# Limpiar y reinstalar
docker stop coffee-api coffee-dashboard
docker rm coffee-api coffee-dashboard
docker rmi coffee-api:latest coffee-dash:latest

# Reconstruir desde cero
docker build --no-cache -f apps/coffee-api/Dockerfile -t coffee-api:latest
```

```
.
docker build --no-cache -f apps/coffee-dash/Dockerfile -t coffee-
dash:latest .
```

Tox:

```
# Limpiar todos los entornos tox
tox run -e clean

# Recrear entornos desde cero
tox run --recreate

# Ejecutar workflow completo de desarrollo
tox run -e checks,test_package,typechecks

# Ver configuración de tox
tox config

# Ejecutar con verbose para debug
tox run -v -e test_package
```

Desarrollo:

```
# Formatear código automáticamente
tox run -e format  # Si está configurado

# Verificar antes de commit
tox run -e checks,test_package

# Entrenar modelo después de cambios
cd apps/coffee-model
tox run -e train
```

📋 Arquitectura del Sistema

```
graph TB
    User[👤 Usuario Final]

    subgraph "🐋 Docker Network: coffee-network"
        subgraph "📊 Coffee Dashboard (Port 5006)"
            Dashboard[🎯 Panel Dashboard]
            DashAPI[🌐 Dashboard API Client]
        end

        subgraph "🔮 Coffee API (Port 8001)"
```

```
            API[⚡ FastAPI Server]
            Predict[🧠 /api/v1/predict]
            Health[❤ /api/v1/health]
        end

        subgraph "🤖 ML Model Package"
            Model[📦 SARIMAX Model]
            Pipeline[⚙ make_prediction()]
        end
    end

    User --> Dashboard
    Dashboard --> DashAPI
    DashAPI -->|HTTP POST| Predict
    DashAPI -->|HTTP GET| Health
    Predict --> Pipeline
    Pipeline --> Model
```

¡Instalación completada! 🎉

Para soporte técnico, contactar al equipo de desarrollo.

Documento generado: $(date) Versión: 1.0