AI-Powered Self-Healing CI/CD Pipeline - Setup Guide

🚀 Quick Start (5 Minutes)

Prerequisites

```bash
# Required software
- Git
- Docker & Docker Compose
- Python 3.9+
- GitHub account
- Kubernetes (minikube for local testing)
```

Step 1: Clone and Setup

```bash
# Create project directory
mkdir ai-healing-cicd
cd ai-healing-cicd

# Create directory structure
mkdir -p pipeline-monitor/app
mkdir -p ml-models/models
mkdir -p demo-app/{tests,app}
mkdir -p kubernetes
mkdir -p monitoring/{grafana/{dashboards,datasources},prometheus}
mkdir -p scripts
mkdir -p nginx
mkdir -p .github/workflows
```

Step 2: Environment Configuration

Create `.env` file in root directory:

```bash
```

```
# .env
GITHUB_TOKEN=ghp_your_token_here
GITHUB_REPO=yourusername/your-repo
SLACK_WEBHOOK_URL=https://hooks.slack.com/services/YOUR/WEBHOOK/URL
DOCKER_USERNAME=your_dockerhub_username
DATABASE_URL=postgresql://cicd_user:secure_password@localhost:5432/cicd_healing
```

Step 3: Get GitHub Personal Access Token

1. Go to GitHub Settings → Developer settings → Personal access tokens
2. Generate new token with permissions:
   - `repo` (Full control)
   - `workflow` (Update workflows)
   - `write:packages` (Upload packages)
3. Copy token to `.env` file

Step 4: Start All Services

bash

```
# Start with Docker Compose
docker-compose up -d

# Check all services are running
docker-compose ps

# View logs
docker-compose logs -f pipeline-monitor
```

Services will be available at:

- **Pipeline Monitor API**: http://localhost:8000
- **Grafana Dashboard**: http://localhost:3000 (admin/admin)
- **Prometheus**: http://localhost:9090
- **Demo App**: http://localhost:8080

Step 5: Initialize Database

bash

```
# Run database initialization
docker-compose exec postgres psql -U cicd_user -d cicd_healing -f /docker-entrypoint-initdb.d/init.sql
```

Step 6: Train ML Models

```
bash

# Train initial models
docker-compose run --rm ml-trainer

# Or manually
python ml-models/train_predictor.py
```

Step 7: Configure GitHub Repository

1. Go to your GitHub repository settings
2. Add the following secrets:
   - `HEALING_API_URL`: Your pipeline monitor URL (ngrok for local testing)
   - `DOCKER_USERNAME`: Docker Hub username
   - `DOCKER_PASSWORD`: Docker Hub password
   - `KUBE_CONFIG`: Your Kubernetes config (for deployment)
   - `SLACK_WEBHOOK`: Slack webhook URL
3. Copy workflow file:

```
bash

cp github-actions/self-healing-pipeline.yml .github/workflows/
git add .github/workflows/
git commit -m "Add self-healing pipeline"
git push
```

📋 Complete File Structure

```
ai-healing-cicd/
├── .env                    # Environment variables
├── .gitignore
├── docker-compose.yml      # All services configuration
```

```
├── README.md
│
├── pipeline-monitor/
│   ├── Dockerfile
│   ├── requirements.txt
│   ├── app/
│   │   ├── __init__.py
│   │   ├── main.py              # FastAPI application
│   │   ├── models.py            # Database models
│   │   ├── github_monitor.py    # GitHub API integration
│   │   ├── healing_engine.py    # Self-healing logic
│   │   └── ml_predictor.py      # ML prediction
│   └── config.yaml
│
├── ml-models/
│   ├── train_predictor.py       # Model training script
│   ├── failure_patterns.py
│   ├── requirements.txt
│   └── models/                  # Saved models directory
│       ├── failure_model.pkl
│       ├── healing_model.pkl
│       └── scaler.pkl
│
├── demo-app/
│   ├── Dockerfile
│   ├── requirements.txt
│   ├── app.py                   # Demo FastAPI app
│   ├── tests/
│   │   ├── test_app.py
│   │   ├── test_api.py
│   │   └── test_integration.py
│   └── README.md
│
├── .github/workflows/
│   └── self-healing-pipeline.yml
│
├── kubernetes/
│   ├── namespace.yaml
│   ├── pipeline-monitor-deployment.yaml
│   ├── postgres-deployment.yaml
│   ├── redis-deployment.yaml
│   ├── grafana-deployment.yaml
│   └── ingress.yaml
│
```

```
├── monitoring/
│   ├── prometheus.yml
│   └── grafana/
│       ├── dashboards/
│       │   └── pipeline-dashboard.json
│       └── datasources/
│           └── prometheus.yaml
│
├── nginx/
│   ├── nginx.conf
│   └── ssl/
│
└── scripts/
    ├── setup.sh              # Complete setup script
    ├── init_db.sql           # Database schema
    ├── deploy.sh             # Deployment script
    └── test_healing.sh       # Test healing scenarios
```

🧪 Demo Scenarios

Scenario 1: Missing Dependency Auto-Fix

bash

```
# 1. Create a branch with missing dependency
git checkout -b test-missing-dep

# 2. Add code that imports a new package
cat << 'EOF' > demo-app/app.py
from fastapi import FastAPI
import pandas as pd  # New dependency not in requirements.txt

app = FastAPI()

@app.get("/")
def read_root():
    df = pd.DataFrame({"test": [1, 2, 3]})
    return {"data": df.to_dict()}
EOF

# 3. Commit and push
git add demo-app/app.py
git commit -m "Add pandas functionality"
git push origin test-missing-dep

# 4. Watch the pipeline:
# - Initial build will FAIL (pandas not in requirements.txt)
# - Healing system detects ModuleNotFoundError
# - Auto-adds "pandas" to requirements.txt
# - Commits the fix
# - Re-triggers workflow
# - Build SUCCEEDS ✅
```

Expected Output:

```
❌ Build failed: ModuleNotFoundError: No module named 'pandas'
🔧 Auto-healing triggered...
✅ Added pandas to requirements.txt
✅ Committed fix: [AUTO-HEAL] Add missing dependency: pandas
🔄 Re-running workflow...
✅ Build successful!
```

Scenario 2: Flaky Test Auto-Fix

bash

```
# 1. Create a flaky test
cat << 'EOF' > demo-app/tests/test_flaky.py
import random
import pytest

def test_sometimes_fails():
    # This test fails 40% of the time
    if random.random() < 0.4:
        assert False, "Randomly failed"
    assert True
EOF

# 2. Push and observe
git add demo-app/tests/test_flaky.py
git commit -m "Add flaky test"
git push

# 3. Healing system will:
# - Detect test failure pattern
# - Add pytest-rerunfailures to requirements.txt
# - Configure pytest with --reruns 3
# - Test passes on retry ✅
```

Scenario 3: Deployment Crash Auto-Rollback

bash

```
# 1. Introduce a critical bug
cat << 'EOF' > demo-app/app.py
from fastapi import FastAPI

app = FastAPI()

@app.get("/")
def read_root():
    # This will crash the application
    raise Exception("Critical bug!")
    return {"status": "ok"}
EOF

# 2. Deploy to production
git add demo-app/app.py
git commit -m "Update endpoint (contains bug)"
git push origin main

# 3. Watch the auto-rollback:
# - Deployment succeeds
# - Health check FAILS
# - Error rate spikes detected
# - Auto-rollback initiated ⚠️
# - Previous version restored ✅
# - Team notified via Slack
```

🎯 Testing the System

Test 1: API Health Check

bash

```
curl http://localhost:8000/
# Expected: {"status":"healthy","timestamp":"..."}
```

Test 2: Manual Healing Trigger

bash

```
curl -X POST http://localhost:8000/manual-heal/12345 \
  -H "Content-Type: application/json"
```

Test 3: Get Analytics

```bash
# Success rate
curl http://localhost:8000/analytics/success-rate

# Failure patterns
curl http://localhost:8000/analytics/failure-patterns

# Recent pipeline status
curl http://localhost:8000/pipelines/status?limit=10
```

Test 4: Trigger Model Retraining

```bash
curl -X POST http://localhost:8000/train-model
```

📊 Grafana Dashboard Setup

1. Open Grafana: http://localhost:3000
2. Login: admin / admin
3. Go to Dashboards → Import
4. Upload `monitoring/grafana/dashboards/pipeline-dashboard.json`

Dashboard Panels:

- Pipeline success rate over time
- Average healing time
- Failure types distribution
- Cost savings metrics
- Auto-heal success rate (Target: 80%+)
- Time saved per week

🔧 Configuration Files

pipeline-monitor/Dockerfile

dockerfile

```
FROM python:3.9-slim

WORKDIR /app

# Install system dependencies
RUN apt-get update && apt-get install -y \
    git \
    curl \
    && rm -rf /var/lib/apt/lists/*

# Copy requirements
COPY requirements.txt .
RUN pip install --no-cache-dir -r requirements.txt

# Copy application
COPY app/ ./app/
COPY ml-models/ ./ml-models/

# Expose port
EXPOSE 8000

# Health check
HEALTHCHECK --interval=30s --timeout=10s --start-period=5s --retries=3 \
    CMD curl -f http://localhost:8000/ || exit 1

# Run application
CMD ["uvicorn", "app.main:app", "--host", "0.0.0.0", "--port", "8000"]
```

pipeline-monitor/requirements.txt

txt

```
fastapi==0.104.1
uvicorn[standard]==0.24.0
pydantic==2.5.0
sqlalchemy==2.0.23
asyncpg==0.29.0
alembic==1.12.1
python-multipart==0.0.6
python-jose[cryptography]==3.3.0
passlib[bcrypt]==1.7.4
python-dotenv==1.0.0
aiofiles==23.2.1
httpx==0.25.1
PyGithub==2.1.1
redis==5.0.1
celery==5.3.4
prometheus-client==0.19.0
scikit-learn==1.3.2
tensorflow==2.15.0
pandas==2.1.3
numpy==1.26.2
joblib==1.3.2
psycopg2-binary==2.9.9
```

scripts/init_db.sql

```sql
```

```sql
-- Database initialization
CREATE TABLE IF NOT EXISTS pipeline_runs (
    id SERIAL PRIMARY KEY,
    run_id VARCHAR(255) UNIQUE NOT NULL,
    repo VARCHAR(255) NOT NULL,
    status VARCHAR(50) NOT NULL,
    conclusion VARCHAR(50),
    started_at TIMESTAMP NOT NULL,
    completed_at TIMESTAMP,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

CREATE TABLE IF NOT EXISTS failure_logs (
    id SERIAL PRIMARY KEY,
    run_id VARCHAR(255) NOT NULL,
    error_type VARCHAR(100) NOT NULL,
    error_message TEXT,
    stack_trace TEXT,
    severity VARCHAR(20),
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    FOREIGN KEY (run_id) REFERENCES pipeline_runs(run_id)
);

CREATE TABLE IF NOT EXISTS healing_actions (
    id SERIAL PRIMARY KEY,
    run_id VARCHAR(255) NOT NULL,
    action_type VARCHAR(100) NOT NULL,
    success BOOLEAN NOT NULL,
    details JSONB,
    changes_made TEXT[],
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    FOREIGN KEY (run_id) REFERENCES pipeline_runs(run_id)
);

CREATE TABLE IF NOT EXISTS metrics (
    id SERIAL PRIMARY KEY,
    run_id VARCHAR(255),
    metric_name VARCHAR(100) NOT NULL,
    metric_value FLOAT NOT NULL,
    timestamp TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

-- Indexes for better performance
```

```
CREATE INDEX idx_pipeline_runs_status ON pipeline_runs(status, conclusion);
CREATE INDEX idx_pipeline_runs_created ON pipeline_runs(created_at DESC);
CREATE INDEX idx_failure_logs_error_type ON failure_logs(error_type);
CREATE INDEX idx_healing_actions_success ON healing_actions(success);

-- Views for analytics
CREATE OR REPLACE VIEW healing_statistics AS
SELECT
  COUNT(*) FILTER (WHERE conclusion = 'failure') as total_failures,
  COUNT(*) FILTER (WHERE EXISTS (
    SELECT 1 FROM healing_actions ha
    WHERE ha.run_id = pipeline_runs.run_id
  )) as healing_attempted,
  COUNT(*) FILTER (WHERE EXISTS (
    SELECT 1 FROM healing_actions ha
    WHERE ha.run_id = pipeline_runs.run_id AND ha.success = true
  )) as healing_successful,
  ROUND(
    100.0 * COUNT(*) FILTER (WHERE EXISTS (
      SELECT 1 FROM healing_actions ha
      WHERE ha.run_id = pipeline_runs.run_id AND ha.success = true
    )) / NULLIF(COUNT(*) FILTER (WHERE conclusion = 'failure'), 0),
    2
  ) as success_rate_percentage
FROM pipeline_runs
WHERE created_at >= NOW() - INTERVAL '30 days';
```

demo-app/app.py

python

```
"""
Demo FastAPI Application
Used to test self-healing capabilities
"""

from fastapi import FastAPI, HTTPException
from pydantic import BaseModel
import logging

logging.basicConfig(level=logging.INFO)
logger = logging.getLogger(__name__)

app = FastAPI(
    title="Demo Application",
    description="Testing self-healing CI/CD pipeline",
    version="1.0.0"
)

class Item(BaseModel):
    name: str
    value: int

@app.get("/")
def read_root():
    """Health check endpoint"""
    return {"status": "healthy", "message": "Demo app is running"}

@app.get("/health")
def health_check():
    """Kubernetes health check"""
    return {"status": "ok"}

@app.post("/items/")
def create_item(item: Item):
    """Create a new item"""
    logger.info(f"Creating item: {item.name}")
    return {"id": 1, "name": item.name, "value": item.value}

@app.get("/items/{item_id}")
def read_item(item_id: int):
    """Get an item by ID"""
    if item_id > 100:
        raise HTTPException(status_code=404, detail="Item not found")
```

```
    return {"id": item_id, "name": f"Item {item_id}"}

if __name__ == "__main__":
    import uvicorn
    uvicorn.run(app, host="0.0.0.0", port=8080)
```

demo-app/tests/test_app.py

```
python
```

```
"""
Test suite for demo application
"""

import pytest
from fastapi.testclient import TestClient
from app import app

client = TestClient(app)

def test_read_root():
    """Test root endpoint"""
    response = client.get("/")
    assert response.status_code == 200
    assert response.json()["status"] == "healthy"

def test_health_check():
    """Test health endpoint"""
    response = client.get("/health")
    assert response.status_code == 200
    assert response.json()["status"] == "ok"

def test_create_item():
    """Test item creation"""
    response = client.post(
        "/items/",
        json={"name": "Test Item", "value": 42}
    )
    assert response.status_code == 200
    data = response.json()
    assert data["name"] == "Test Item"
    assert data["value"] == 42

def test_read_item():
    """Test reading an item"""
    response = client.get("/items/1")
    assert response.status_code == 200
    assert response.json()["id"] == 1

def test_read_nonexistent_item():
    """Test reading a non-existent item"""
    response = client.get("/items/999")
    assert response.status_code == 404
```

```python
@pytest.mark.timeout(10)
def test_performance():
    """Test response time"""
    import time
    start = time.time()
    response = client.get("/")
    duration = time.time() - start
    assert duration < 1.0  # Should respond in less than 1 second
    assert response.status_code == 200
```

🚢 Kubernetes Deployment

kubernetes/namespace.yaml

yaml

```yaml
apiVersion: v1
kind: Namespace
metadata:
  name: healing-cicd
  labels:
    name: healing-cicd
```

kubernetes/pipeline-monitor-deployment.yaml

yaml

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: pipeline-monitor
  namespace: healing-cicd
  labels:
    app: pipeline-monitor
spec:
  replicas: 2
  selector:
    matchLabels:
      app: pipeline-monitor
  template:
    metadata:
      labels:
        app: pipeline-monitor
    spec:
      containers:
      - name: pipeline-monitor
        image: your-registry/pipeline-monitor:latest
        ports:
        - containerPort: 8000
        env:
        - name: DATABASE_URL
          valueFrom:
            secretKeyRef:
              name: db-credentials
              key: url
        - name: GITHUB_TOKEN
          valueFrom:
            secretKeyRef:
              name: github-credentials
              key: token
        - name: REDIS_URL
          value: redis://redis:6379/0
        resources:
          requests:
            memory: "256Mi"
            cpu: "250m"
          limits:
            memory: "512Mi"
            cpu: "500m"
        livenessProbe:
```

```
        httpGet:
          path: /
          port: 8000
        initialDelaySeconds: 30
        periodSeconds: 10
      readinessProbe:
        httpGet:
          path: /
          port: 8000
        initialDelaySeconds: 5
        periodSeconds: 5
---
apiVersion: v1
kind: Service
metadata:
  name: pipeline-monitor
  namespace: healing-cicd
spec:
  selector:
    app: pipeline-monitor
  ports:
  - protocol: TCP
    port: 8000
    targetPort: 8000
  type: LoadBalancer
```

📈 Performance Metrics & Expected Results

Target Metrics (As Per Abstract)

Metric	Target	How to Measure
Pipeline Time Reduction	25-35%	Compare before/after average build times
Auto-Healing Success Rate	80%+	`SELECT success_rate_percentage FROM healing_statistics`
Cost Savings	Significant	Track cloud resource usage reduction
Developer Productivity	Enhanced	Reduced time spent on manual fixes

Measuring Results

```bash
```

```
# Get current statistics
curl http://localhost:8000/analytics/success-rate

# Expected output:
{
  "total_failures": 100,
  "healing_attempted": 95,
  "healing_successful": 85,
  "success_rate": 89.47,
  "time_saved_minutes": 425,
  "cost_saved_usd": 127.50
}
```

Calculation Methods

Time Saved:

```
python

# Average manual fix time: 15 minutes
# Successful auto-heals: 85
# Time saved = 85 × 15 = 1,275 minutes (21.25 hours)
```

Cost Saved:

```
python

# Average cloud cost per failed pipeline: $2.50
# Prevented re-runs: 85
# Cost saved = 85 × $1.50 = $127.50
```

Pipeline Time Reduction:

```
python

# Before: Average 12 minutes per build
# After: Average 8.5 minutes per build
# Reduction: (12 - 8.5) / 12 × 100 = 29.2%
```

🎓 Project Presentation Materials

Key Points for Defense

1. **Problem Statement:**
   - CI/CD pipelines fail frequently
   - Manual fixes waste developer time
   - Cloud resources wasted on failed builds
2. **Solution:**
   - AI-powered failure prediction
   - Automated healing mechanisms
   - Intelligent rollback system
3. **Technical Implementation:**
   - FastAPI backend with ML integration
   - Scikit-learn & TensorFlow for predictions
   - GitHub Actions integration
   - Kubernetes orchestration
   - Grafana monitoring
4. **Key Achievements:**
   - 80%+ auto-healing success rate
   - 25-35% pipeline time reduction
   - Real-time failure detection
   - Automated dependency management
   - Smart deployment rollback
5. **Demo Scenarios:**
   - Show live missing dependency fix
   - Demonstrate flaky test handling
   - Show deployment rollback
   - Display Grafana metrics

Presentation Structure

```
1. Introduction (2 min)
   - Problem overview
   - Statistics on CI/CD failures

2. System Architecture (3 min)
   - Component diagram
   - Data flow
   - Technology stack

3. Core Features (5 min)
   - Failure prediction
   - Auto-healing mechanisms
   - Monitoring dashboard

4. Live Demo (5 min)
   - Trigger failures
   - Show auto-healing
   - Display metrics

5. Results & Metrics (2 min)
   - Success rate
   - Time/cost savings
   - Developer productivity

6. Future Enhancements (1 min)
   - Advanced ML models
   - More healing strategies
   - Multi-cloud support

7. Q&A (2 min)
```

🔍 Troubleshooting

Issue: Services won't start

```bash
```

```
# Check Docker status
docker-compose ps

# View logs
docker-compose logs pipeline-monitor

# Restart services
docker-compose restart

# Complete reset
docker-compose down -v
docker-compose up -d
```

Issue: Database connection error

```bash
# Check PostgreSQL is running
docker-compose exec postgres pg_isready -U cicd_user

# Reset database
docker-compose down postgres
docker-compose up -d postgres
```

Issue: GitHub API rate limit

```bash
# Check rate limit status
curl -H "Authorization: token $GITHUB_TOKEN" \
  https://api.github.com/rate_limit
```

Issue: ML model not loading

```bash
# Retrain models
docker-compose run --rm ml-trainer

# Check model files
ls -la ml-models/models/
```

📝 Next Steps

1. **Week 1-2:** Setup infrastructure and basic components
2. **Week 3-4:** Implement healing strategies
3. **Week 5-6:** Train ML models with real data
4. **Week 7:** Integration testing
5. **Week 8:** Documentation and presentation prep

🎯 Success Criteria Checklist

- [ ] All services running without errors
- [ ] GitHub Actions workflow integrated
- [ ] At least 3 demo scenarios working
- [ ] ML models trained and predicting
- [ ] Grafana dashboard showing metrics
- [ ] 80%+ healing success rate achieved
- [ ] Documentation complete
- [ ] Presentation ready
- [ ] Video demo recorded

📚 Additional Resources

- FastAPI Documentation: https://fastapi.tiangolo.com
- GitHub Actions: https://docs.github.com/en/actions
- Kubernetes: https://kubernetes.io/docs
- Scikit-learn: https://scikit-learn.org
- Docker Compose: https://docs.docker.com/compose

🎉 You're All Set!

Your AI-Powered Self-Healing CI/CD Pipeline is ready to go. Start by running the demo scenarios and customizing the healing strategies for your needs.

For questions or issues, refer to the troubleshooting section or check the logs.

Good luck with your final year project! 🚀