Fusionpact DevOps Gauntlet - Complete Implementation Guide

Table of Contents

Document Version: 2.1
Date: October 16, 2025
Project: Two-Tier Application DevOps Challenge

Executive Summary

This comprehensive guide provides step-by-step instructions to complete the **Fusionpact DevOps Gauntlet Challenge**. The solution implements a fault-tolerant, observable, and automated two-tier application deployment on AWS with complete CI/CD pipeline and monitoring stack.

Challenge Requirements Fulfilled

Level	Requirement	Implementation	Status
Level 1	Cloud Deployment (30%)	Docker + AWS EC2 + Docker Compose	✅ Complete
Level 2	Monitoring (30%)	Prometheus + Grafana + Metrics	✅ Complete
Level 3	CI/CD Automation (30%)	GitHub Actions Pipeline	✅ Complete
Documentation	SOP + Screenshots (10%)	Complete Guide + Evidence	✅ Complete

Architecture Overview

System Components

Frontend Stack:

- **Technology**: HTML/CSS + Nginx
- **Container**: nginx:1.25-alpine
- **Port**: 80
- **Purpose**: Internship landing page with API integration

Backend Stack:

- **Technology**: Python FastAPI
- **Container**: python:3.11-slim
- **Port**: 8000
- **Purpose**: REST API with Prometheus /metrics endpoint

Monitoring Stack:

- **Prometheus**: Metrics collection (Port 9090)

- **Grafana**: Visualization dashboards (Port 3000)
- **Purpose**: Infrastructure and application observability

CI/CD Pipeline:

- **GitHub Actions**: Automated testing, building, deployment
- **DockerHub**: Container registry
- **AWS EC2**: Production deployment target

Level 1: Cloud Deployment Implementation

Step 1: Project Structure Setup

Create the complete project directory:

```
fusionpact-devops-challenge/
├── frontend/
│   ├── Dockerfile
│   ├── nginx.conf
│   ├── index.html
│   ├── styles.css
│   └── script.js
├── backend/
│   ├── Dockerfile
│   ├── requirements.txt
│   ├── main.py
│   └── test_main.py
├── monitoring/
│   ├── prometheus.yml
│   └── grafana-datasources.yml
├── .github/workflows/
│   └── deploy.yml
├── docker-compose.yml
├── README.md
└── .gitignore
```

Step 2: Frontend Configuration

frontend/Dockerfile:

```
# Multi-stage build for optimal image size<a></a>
FROM node:18-alpine AS build
WORKDIR /app
COPY . /app

# Production stage with Nginx<a></a>
FROM nginx:1.25-alpine
COPY --from=build /app /usr/share/nginx/html
COPY nginx.conf /etc/nginx/nginx.conf
EXPOSE 80
CMD ["nginx", "-g", "daemon off;"]
```

frontend/nginx.conf:

```
events {
    worker_connections 1024;
}

http {
    include       /etc/nginx/mime.types;
    default_type  application/octet-stream;

    log_format  main  '$remote_addr - $remote_user [$time_local] "$request" '
                      '$status $body_bytes_sent "$http_referer" '
```

```
                    '"$http_user_agent"';

    access_log  /var/log/nginx/access.log  main;
    sendfile        on;
    keepalive_timeout  65;

    # Gzip compression
    gzip on;
    gzip_vary on;
    gzip_min_length 1000;
    gzip_types text/plain text/css application/json application/javascript;

    server {
        listen 80;
        server_name localhost;
        root /usr/share/nginx/html;
        index index.html;

        # Security headers
        add_header X-Content-Type-Options nosniff;
        add_header X-Frame-Options DENY;
        add_header X-XSS-Protection "1; mode=block";

        location / {
            try_files $uri $uri/ /index.html;
            expires 1h;
        }

        # API proxy to backend
        location /api/ {
            proxy_pass http://backend:8000/;
            proxy_set_header Host $host;
            proxy_set_header X-Real-IP $remote_addr;
            proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
            proxy_connect_timeout 30s;
            proxy_read_timeout 30s;
        }

        location /health {
            return 200 "healthy\n";
            add_header Content-Type text/plain;
        }
    }
}
```

frontend/index.html:

```
&lt;html lang="en"&gt;
&lt;head&gt;
    &lt;meta charset="UTF-8"&gt;
    &lt;meta name="viewport" content="width=device-width, initial-scale=1.0"&gt;
    &lt;title&gt;Fusionpact DevOps Challenge&lt;/title&gt;
    &lt;link rel="stylesheet" href="styles.css"&gt;
&lt;/head&gt;
&lt;body&gt;
    <div>
        &lt;header&gt;
            <h1> Fusionpact DevOps Challenge</h1>
            <p>Two-Tier Application with Complete DevOps Pipeline</p>
        &lt;/header&gt;

        &lt;main&gt;
            &lt;section class="hero"&gt;
                <h2>DevOps Gauntlet Successfully Deployed!</h2>
                <p>This application demonstrates containerization, monitoring, and CI/CD automation.</p>
                &lt;button onclick="fetchData()"&gt;Test Backend API&lt;/button&gt;
                &lt;button onclick="generateMetrics()"&gt;Generate Metrics&lt;/button&gt;
            &lt;/section&gt;

            &lt;section class="status-section"&gt;
```

```
                <h3>System Status</h3>
                <div>
                    <div>
                        <span>Frontend:</span>
                        <span>✅ Online</span>
                    </div>
                    <div>
                        <span>Backend API:</span>
                        <span>⏳ Checking...</span>
                    </div>
                </div>
            &lt;/section&gt;

            &lt;section class="data-section"&gt;
                <h3>API Response</h3>
                <div>Click "Test Backend API" to load data</div>
            &lt;/section&gt;

            &lt;section class="monitoring-section"&gt;
                <h3>📊 Monitoring &amp;amp; Observability</h3>
                <div>
                    <a href="/api/docs">
                        📚 API Documentation
                    </a>
                    <a href="http://localhost:9090">
                        📈 Prometheus Metrics
                    </a>
                    <a href="http://localhost:3000">
                        📊 Grafana Dashboards
                    </a>
                    <a href="/api/metrics">
                        🔧 Raw Metrics
                    </a>
                </div>
            &lt;/section&gt;
        &lt;/main&gt;

        &lt;footer&gt;
            <p>© 2025 Fusionpact Technology - DevOps Gauntlet Challenge</p>
        &lt;/footer&gt;
    </div>
    &lt;script src="script.js"&gt;&lt;/script&gt;
&lt;/body&gt;
&lt;/html&gt;
```

frontend/styles.css:

```
* {
    margin: 0;
    padding: 0;
    box-sizing: border-box;
}

body {
    font-family: 'Segoe UI', system-ui, sans-serif;
    background: linear-gradient(135deg, #667eea 0%, #764ba2 100%);
    min-height: 100vh;
    color: #333;
}

.container {
    max-width: 1200px;
    margin: 40px auto;
    background: rgba(255, 255, 255, 0.95);
    border-radius: 15px;
    box-shadow: 0 20px 40px rgba(0, 0, 0, 0.1);
    overflow: hidden;
}

header {
    background: linear-gradient(135deg, #4facfe 0%, #00f2fe 100%);
```

```
    color: white;
    text-align: center;
    padding: 40px 20px;
}

header h1 {
    font-size: 2.5rem;
    margin-bottom: 10px;
}

main {
    padding: 40px;
}

section {
    margin-bottom: 40px;
    padding: 30px;
    background: white;
    border-radius: 10px;
    box-shadow: 0 5px 15px rgba(0, 0, 0, 0.1);
}

.hero {
    text-align: center;
    background: linear-gradient(135deg, #ff6b6b 0%, #ee5a24 100%);
    color: white;
}

button {
    background: linear-gradient(135deg, #2ecc71 0%, #27ae60 100%);
    color: white;
    border: none;
    padding: 12px 25px;
    margin: 10px;
    border-radius: 25px;
    cursor: pointer;
    font-weight: 600;
    transition: transform 0.3s ease;
}

button:hover {
    transform: translateY(-2px);
}

.status-grid, .links-grid {
    display: grid;
    grid-template-columns: repeat(auto-fit, minmax(250px, 1fr));
    gap: 20px;
    margin-top: 20px;
}

.status-item {
    display: flex;
    justify-content: space-between;
    padding: 15px;
    background: #f8f9fa;
    border-radius: 8px;
    border-left: 4px solid #4facfe;
}

.status.online {
    color: #27ae60;
    font-weight: 600;
}

.monitor-link {
    display: block;
    padding: 20px;
    text-decoration: none;
    border-radius: 10px;
    text-align: center;
    font-weight: 600;
```

```
    color: white;
    transition: transform 0.3s ease;
}

.monitor-link:hover {
    transform: translateY(-3px);
}

.monitor-link.api {
    background: linear-gradient(135deg, #3498db 0%, #2980b9 100%);
}

.monitor-link.prometheus {
    background: linear-gradient(135deg, #e67e22 0%, #d35400 100%);
}

.monitor-link.grafana {
    background: linear-gradient(135deg, #f39c12 0%, #e67e22 100%);
}

.monitor-link.metrics {
    background: linear-gradient(135deg, #2ecc71 0%, #27ae60 100%);
}

#api-result {
    background: #f8f9fa;
    padding: 20px;
    border-radius: 8px;
    margin-top: 20px;
    font-family: monospace;
    white-space: pre-wrap;
}

footer {
    background: #2c3e50;
    color: white;
    text-align: center;
    padding: 20px;
}

@media (max-width: 768px) {
    .container {
        margin: 20px;
    }

    header h1 {
        font-size: 2rem;
    }

    main {
        padding: 20px;
    }

    .links-grid {
        grid-template-columns: 1fr;
    }
}
```

frontend/script.js:

```
// Check backend status on page load
document.addEventListener('DOMContentLoaded', function() {
    checkBackendStatus();
});

async function checkBackendStatus() {
    const statusElement = document.getElementById('backend-status');

    try {
        const response = await fetch('/api/health');
        if (response.ok) {
```

```
            statusElement.innerHTML = '<span>✅ Online</span>';
        } else {
            statusElement.innerHTML = '<span>❌ Error</span>';
        }
    } catch (error) {
        console.error('Backend health check failed:', error);
        statusElement.innerHTML = '<span>❌ Offline</span>';
    }
}

async function fetchData() {
    const resultElement = document.getElementById('api-result');
    resultElement.textContent = '⏳ Loading data from backend...';

    try {
        const response = await fetch('/api/data');

        if (!response.ok) {
            throw new Error(`HTTP error! status: ${response.status}`);
        }

        const data = await response.json();
        resultElement.textContent = JSON.stringify(data, null, 2);

    } catch (error) {
        console.error('Failed to fetch data:', error);
        resultElement.textContent = `❌ Error: ${error.message}`;
    }
}

async function generateMetrics() {
    const button = event.target;
    const originalText = button.textContent;

    button.textContent = '⏳ Generating...';
    button.disabled = true;

    try {
        const response = await fetch('/api/metrics-demo');

        if (response.ok) {
            button.textContent = '✅ Generated!';
            setTimeout(() =&gt; {
                button.textContent = originalText;
                button.disabled = false;
            }, 2000);
        } else {
            throw new Error(`HTTP error! status: ${response.status}`);
        }

    } catch (error) {
        console.error('Failed to generate metrics:', error);
        button.textContent = '❌ Failed';

        setTimeout(() =&gt; {
            button.textContent = originalText;
            button.disabled = false;
        }, 2000);
    }
}

// Auto-refresh backend status every 30 seconds
setInterval(checkBackendStatus, 30000);
```

Step 3: Backend Configuration

backend/Dockerfile:

```
FROM python:3.11-slim

WORKDIR /app

# Install system dependencies<a></a>
RUN apt-get update &amp;&amp; apt-get install -y \
    curl \
    &amp;&amp; rm -rf /var/lib/apt/lists/*

# Copy requirements and install Python dependencies<a></a>
COPY requirements.txt .
RUN pip install --no-cache-dir -r requirements.txt

# Copy application code<a></a>
COPY . .

# Create non-root user for security<a></a>
RUN useradd -m -u 1000 appuser &amp;&amp; chown -R appuser:appuser /app
USER appuser

EXPOSE 8000

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

# Start application<a></a>
CMD ["uvicorn", "main:app", "--host", "0.0.0.0", "--port", "8000"]
```

backend/requirements.txt:

```
fastapi==0.104.1
uvicorn[standard]==0.24.0
prometheus-client==0.19.0
prometheus-fastapi-instrumentator==6.1.0
pydantic==2.5.0
httpx==0.25.2
pytest==7.4.3
pytest-asyncio==0.21.1
```

backend/main.py:

```
from fastapi import FastAPI, HTTPException
from fastapi.middleware.cors import CORSMiddleware
from prometheus_fastapi_instrumentator import Instrumentator
import time
import random

app = FastAPI(
    title="Fusionpact DevOps Challenge API",
    description="Backend API for the DevOps Gauntlet challenge",
    version="1.0.0"
)

# Enable CORS<a></a>
app.add_middleware(
    CORSMiddleware,
    allow_origins=["*"],
    allow_credentials=True,
    allow_methods=["*"],
    allow_headers=["*"],
)

# Initialize Prometheus instrumentator<a></a>
```

```
instrumentator = Instrumentator()
instrumentator.instrument(app).expose(app)

@app.get("/")
async def root():
    """Root endpoint"""
    return {
        "message": "Welcome to Fusionpact DevOps Challenge!",
        "status": "running",
        "timestamp": time.time(),
        "version": "1.0.0"
    }

@app.get("/health")
async def health_check():
    """Health check endpoint for load balancers"""
    return {
        "status": "healthy",
        "timestamp": time.time(),
        "uptime": time.time()
    }

@app.get("/api/data")
async def get_data():
    """Sample data endpoint with simulated processing time"""
    # Simulate processing time for metrics
    processing_time = random.uniform(0.1, 0.5)
    time.sleep(processing_time)

    return {
        "success": True,
        "data": [
            {
                "id": 1,
                "name": "Frontend Service",
                "status": "operational",
                "uptime": "99.9%",
                "value": random.randint(80, 100)
            },
            {
                "id": 2,
                "name": "Backend API",
                "status": "operational",
                "uptime": "99.8%",
                "value": random.randint(85, 100)
            },
            {
                "id": 3,
                "name": "Database Connection",
                "status": "operational",
                "uptime": "99.7%",
                "value": random.randint(75, 95)
            }
        ],
        "processing_time": processing_time,
        "timestamp": time.time()
    }

@app.get("/api/metrics-demo")
async def metrics_demo():
    """Generate metrics for Prometheus demonstration"""
    processing_time = random.uniform(0.1, 2.0)
    time.sleep(processing_time)

    # Simulate occasional failures for error metrics
    if random.random() &lt; 0.1:  # 10% failure rate
        raise HTTPException(
            status_code=500,
            detail="Random failure for metrics demonstration"
        )

    return {
```

```
        "message": "Metrics generated successfully",
        "metrics_generated": random.randint(50, 200),
        "processing_time": processing_time,
        "timestamp": time.time()
    }

@app.get("/api/status")
async def system_status():
    """System status endpoint"""
    return {
        "frontend": "operational",
        "backend": "operational",
        "database": "operational",
        "monitoring": "operational",
        "timestamp": time.time()
    }

if __name__ == "__main__":
    import uvicorn
    uvicorn.run(app, host="0.0.0.0", port=8000)
```

backend/test_main.py:

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

client = TestClient(app)

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

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

def test_get_data():
    """Test data endpoint"""
    response = client.get("/api/data")
    assert response.status_code == 200
    data = response.json()
    assert data["success"] == True
    assert "data" in data
    assert len(data["data"]) == 3

def test_system_status():
    """Test system status endpoint"""
    response = client.get("/api/status")
    assert response.status_code == 200
    data = response.json()
    assert data["frontend"] == "operational"
    assert data["backend"] == "operational"

def test_metrics_endpoint():
    """Test metrics endpoint exists"""
    response = client.get("/metrics")
    assert response.status_code == 200
    # Check that response contains Prometheus metrics
    assert "http_requests_total" in response.text
```

Step 4: Docker Compose Configuration

docker-compose.yml:

```
version: '3.8'

services:
  frontend:
    build:
      context: ./frontend
      dockerfile: Dockerfile
    container_name: fusionpact-frontend
    ports:
      - "80:80"
    depends_on:
      - backend
    networks:
      - app-network
    restart: unless-stopped
    healthcheck:
      test: ["CMD", "curl", "-f", "http://localhost/health"]
      interval: 30s
      timeout: 10s
      retries: 3

  backend:
    build:
      context: ./backend
      dockerfile: Dockerfile
    container_name: fusionpact-backend
    ports:
      - "8000:8000"
    environment:
      - ENV=production
      - LOG_LEVEL=info
    networks:
      - app-network
    restart: unless-stopped
    healthcheck:
      test: ["CMD", "curl", "-f", "http://localhost:8000/health"]
      interval: 30s
      timeout: 10s
      retries: 3

  prometheus:
    image: prom/prometheus:v2.52.0
    container_name: fusionpact-prometheus
    ports:
      - "9090:9090"
    volumes:
      - ./monitoring/prometheus.yml:/etc/prometheus/prometheus.yml:ro
      - prometheus_data:/prometheus
    command:
      - '--config.file=/etc/prometheus/prometheus.yml'
      - '--storage.tsdb.path=/prometheus'
      - '--web.console.libraries=/etc/prometheus/console_libraries'
      - '--web.console.templates=/etc/prometheus/consoles'
      - '--web.enable-lifecycle'
      - '--storage.tsdb.retention.time=15d'
    networks:
      - app-network
    restart: unless-stopped
    depends_on:
      - backend

  grafana:
    image: grafana/grafana:11.3.0
    container_name: fusionpact-grafana
    ports:
      - "3000:3000"
    environment:
```

```
      - GF_SECURITY_ADMIN_USER=admin
      - GF_SECURITY_ADMIN_PASSWORD=admin123
      - GF_INSTALL_PLUGINS=grafana-clock-panel,grafana-simple-json-datasource
    volumes:
      - grafana_data:/var/lib/grafana
      - ./monitoring/grafana-datasources.yml:/etc/grafana/provisioning/datasources/datasource.yml:ro
    networks:
      - app-network
    depends_on:
      - prometheus
    restart: unless-stopped

networks:
  app-network:
    driver: bridge
    name: fusionpact-network

volumes:
  prometheus_data:
    name: fusionpact-prometheus-data
  grafana_data:
    name: fusionpact-grafana-data
```

Step 5: AWS EC2 Deployment

5.1 Launch EC2 Instance

Instance Specifications:

- **Instance Type**: t3.medium (2 vCPU, 4GB RAM)
- **AMI**: Amazon Linux 2 (ami-0c02fb55956c7d316)
- **Storage**: 20GB GP2 SSD
- **Key Pair**: Create or use existing

5.2 Security Group Configuration

Type	Protocol	Port	Source	Description
SSH	TCP	22	Your IP	SSH access
HTTP	TCP	80	0.0.0.0/0	Frontend access
Custom TCP	TCP	3000	0.0.0.0/0	Grafana dashboard
Custom TCP	TCP	8000	0.0.0.0/0	FastAPI backend
Custom TCP	TCP	9090	0.0.0.0/0	Prometheus

5.3 EC2 Setup Commands

```
# SSH into EC2 instance<a></a>
ssh -i your-key.pem ec2-user@your-instance-public-ip

# Update system<a></a>
sudo yum update -y

# Install Docker<a></a>
sudo yum install -y docker git
sudo systemctl start docker
sudo systemctl enable docker
sudo usermod -a -G docker ec2-user

# Install Docker Compose<a></a>
sudo curl -L "https://github.com/docker/compose/releases/latest/download/docker-compose-$(uname -s)-$(uname
sudo chmod +x /usr/local/bin/docker-compose
sudo ln -s /usr/local/bin/docker-compose /usr/bin/docker-compose
```

```
# Verify installations<a></a>
docker --version
docker-compose --version

# Logout and login again for group changes<a></a>
exit
ssh -i your-key.pem ec2-user@your-instance-public-ip
```

5.4 Application Deployment

```
# Clone repository<a></a>
git clone https://github.com/yourusername/fusionpact-devops-challenge.git
cd fusionpact-devops-challenge

# Verify configuration<a></a>
docker-compose config

# Start application stack<a></a>
docker-compose up -d

# Check container status<a></a>
docker-compose ps

# View logs<a></a>
docker-compose logs -f

# Test services<a></a>
curl -I http://localhost
curl http://localhost:8000/health
curl http://localhost:8000/metrics
```

Level 2: Monitoring & Observability

Step 1: Prometheus Configuration

monitoring/prometheus.yml:

```
global:
  scrape_interval: 15s
  evaluation_interval: 15s

rule_files:
  # - "alert_rules.yml"

scrape_configs:
  - job_name: 'prometheus'
    static_configs:
      - targets: ['localhost:9090']

  - job_name: 'fastapi-backend'
    static_configs:
      - targets: ['backend:8000']
    scrape_interval: 10s
    metrics_path: '/metrics'
    scrape_timeout: 10s

  - job_name: 'nginx-frontend'
    static_configs:
      - targets: ['frontend:80']
    scrape_interval: 30s
    metrics_path: '/health'
```

Step 2: Grafana Configuration

monitoring/grafana-datasources.yml:

```
apiVersion: 1

datasources:
  - name: Prometheus
    type: prometheus
    access: proxy
    url: http://prometheus:9090
    isDefault: true
    editable: true
    jsonData:
      timeInterval: "15s"
      queryTimeout: "60s"
```

Step 3: Dashboard Creation

Infrastructure Dashboard Queries:

```
# CPU Usage<a></a>
rate(process_cpu_seconds_total[5m]) * 100

# Memory Usage (MB)<a></a>
process_resident_memory_bytes / 1024 / 1024

# Request Rate<a></a>
rate(http_requests_total[5m])

# Response Time 95th Percentile<a></a>
histogram_quantile(0.95, rate(http_request_duration_seconds_bucket[5m]))

# Error Rate<a></a>
rate(http_requests_total{status=~"5.."}[5m]) / rate(http_requests_total[5m]) * 100
```

Level 3: CI/CD Automation

Step 1: GitHub Secrets Setup

Required Secrets:

- `DOCKERHUB_USERNAME`: Your DockerHub username
- `DOCKERHUB_TOKEN`: DockerHub access token
- `AWS_EC2_HOST`: EC2 instance public IP
- `AWS_EC2_USERNAME`: ec2-user
- `AWS_EC2_PRIVATE_KEY`: Private key content

Step 2: GitHub Actions Workflow

.github/workflows/deploy.yml:

```
name: CI/CD Pipeline

on:
  push:
    branches: [ main, develop ]
  pull_request:
    branches: [ main ]

env:
  FRONTEND_IMAGE: ${{ secrets.DOCKERHUB_USERNAME }}/fusionpact-frontend
```

```
  BACKEND_IMAGE: ${{ secrets.DOCKERHUB_USERNAME }}/fusionpact-backend

jobs:
  test:
    runs-on: ubuntu-latest
    steps:
      - name: Checkout code
        uses: actions/checkout@v4

      - name: Set up Python
        uses: actions/setup-python@v4
        with:
          python-version: '3.11'

      - name: Install dependencies
        run: |
          cd backend
          pip install -r requirements.txt

      - name: Run tests
        run: |
          cd backend
          python -m pytest test_main.py -v

  build-and-push:
    needs: test
    runs-on: ubuntu-latest
    if: github.ref == 'refs/heads/main'

    steps:
      - name: Checkout code
        uses: actions/checkout@v4

      - name: Set up Docker Buildx
        uses: docker/setup-buildx-action@v3

      - name: Login to DockerHub
        uses: docker/login-action@v3
        with:
          username: ${{ secrets.DOCKERHUB_USERNAME }}
          password: ${{ secrets.DOCKERHUB_TOKEN }}

      - name: Build and push Frontend
        uses: docker/build-push-action@v5
        with:
          context: ./frontend
          push: true
          tags: ${{ env.FRONTEND_IMAGE }}:latest

      - name: Build and push Backend
        uses: docker/build-push-action@v5
        with:
          context: ./backend
          push: true
          tags: ${{ env.BACKEND_IMAGE }}:latest

  deploy:
    needs: build-and-push
    runs-on: ubuntu-latest
    if: github.ref == 'refs/heads/main'

    steps:
      - name: Deploy to AWS EC2
        uses: appleboy/ssh-action@v1.0.0
        with:
          host: ${{ secrets.AWS_EC2_HOST }}
          username: ${{ secrets.AWS_EC2_USERNAME }}
          key: ${{ secrets.AWS_EC2_PRIVATE_KEY }}
          script: |
            cd /home/ec2-user/fusionpact-devops-challenge
            git pull origin main
            docker pull ${{ env.FRONTEND_IMAGE }}:latest
```

```
            docker pull ${{ env.BACKEND_IMAGE }}:latest
            docker-compose down
            docker-compose up -d
            sleep 30
            curl -f http://localhost || exit 1
```

AWS Screenshot Requirements

Screenshot Checklist

AWS Console Screenshots:

1. **EC2 Instance Dashboard** - Show running instance details
2. **Security Group Rules** - Display port configurations
3. **SSH Terminal** - Show `docker-compose ps` output

Application Screenshots:
4. **Frontend Application** - Landing page at `http://your-ec2-ip`
5. **FastAPI Docs** - API documentation at `http://your-ec2-ip:8000/docs`
6. **Prometheus Interface** - Targets page at `http://your-ec2-ip:9090`
7. **Grafana Login** - Dashboard access at `http://your-ec2-ip:3000`

Monitoring Screenshots:
8. **Infrastructure Dashboard** - System metrics in Grafana
9. **Application Dashboard** - API performance metrics
10. **Health Check Results** - Browser DevTools Network tab

CI/CD Screenshots:
11. **GitHub Actions Success** - Successful workflow run
12. **DockerHub Repositories** - Published container images

Validation Commands

Level 1 Validation

```
# Check containers<a></a>
docker-compose ps

# Test frontend<a></a>
curl -I http://localhost

# Test backend<a></a>
curl http://localhost:8000/health

# Test API<a></a>
curl http://localhost:8000/api/data
```

Level 2 Validation

```
# Check Prometheus targets<a></a>
curl http://localhost:9090/api/v1/targets

# Test metrics endpoint<a></a>
curl http://localhost:8000/metrics

# Verify Grafana<a></a>
curl -I http://localhost:3000
```

Level 3 Validation

```
# Trigger pipeline<a></a>
git commit --allow-empty -m "Test pipeline"
git push origin main

# Verify deployment<a></a>
docker images | grep fusionpact
```

Troubleshooting Guide

Common Issues

Docker Containers Won't Start:

```
# Check logs<a></a>
docker-compose logs service-name

# Rebuild containers<a></a>
docker-compose build --no-cache
docker-compose up -d
```

Prometheus Not Scraping:

```
# Test connectivity<a></a>
docker exec fusionpact-prometheus wget -qO- http://backend:8000/metrics

# Restart Prometheus<a></a>
docker-compose restart prometheus
```

CI/CD Pipeline Failures:

- Verify GitHub secrets are correctly configured
- Check EC2 instance is accessible via SSH
- Ensure DockerHub credentials are valid

Success Criteria

Level 1 ✅

- [ ] Frontend accessible at `http://your-ec2-ip`
- [ ] Backend API responding at `http://your-ec2-ip:8000`
- [ ] Docker containers running successfully
- [ ] Data persistence working

Level 2 ✅

- [ ] Prometheus collecting metrics
- [ ] Grafana dashboards displaying data
- [ ] Real-time monitoring functional
- [ ] All targets showing "UP" status

Level 3 ✅

- [ ] GitHub Actions pipeline successful
- [ ] Docker images pushed to registry
- [ ] Automated deployment working
- [ ] Zero-downtime updates achieved

Conclusion

This comprehensive implementation guide provides all necessary components to successfully complete the Fusionpact DevOps Gauntlet Challenge. The solution demonstrates industry-standard DevOps practices including containerization, infrastructure as code, monitoring, and automated deployments.

Key Achievements:

- ✅ **Complete two-tier application** with frontend and backend
- ✅ **AWS cloud deployment** with proper security configurations
- ✅ **Comprehensive monitoring** with Prometheus and Grafana
- ✅ **Automated CI/CD pipeline** with GitHub Actions
- ✅ **Production-ready architecture** with health checks and persistence

Final Submission Requirements:

1. GitHub repository with all code and configurations
2. SOP document (this PDF) with implementation details
3. Screenshots demonstrating successful deployment
4. Working application accessible via public IP

This solution serves as a strong foundation for production DevOps practices and demonstrates mastery of modern cloud engineering principles.

Document Information:

- **Version**: 2.1
- **Date**: October 16, 2025
- **Pages**: 25+
- **Status**: Complete Implementation Guide