Production-Optimized Dockerfiles

These Dockerfiles are optimized for production deployment with security, performance, and size considerations.

Directory Structure

Create a `docker/` directory in your project root:

```
threatcompass/
├── docker/
│   ├── Dockerfile.flask-app
│   ├── Dockerfile.celery-worker
│   ├── Dockerfile.celery-beat
│   └── requirements/
│       ├── base.txt
│       ├── production.txt
│       └── development.txt
├── requirements.txt -> docker/requirements/production.txt
└── requirements-dev.txt -> docker/requirements/development.txt
```

Flask Application Dockerfile

dockerfile

```dockerfile
# docker/Dockerfile.flask-app
# Production-optimized Flask application container

# Use Python 3.11 slim image for smaller size
FROM python:3.11-slim as base

# Set environment variables
ENV PYTHONUNBUFFERED=1 \
    PYTHONDONTWRITEBYTECODE=1 \
    PIP_NO_CACHE_DIR=1 \
    PIP_DISABLE_PIP_VERSION_CHECK=1 \
    PIP_DEFAULT_TIMEOUT=100

# Install system dependencies
RUN apt-get update && apt-get install -y --no-install-recommends \
    # Build dependencies
    gcc \
    g++ \
    # PostgreSQL client library
    libpq-dev \
    # For health checks
    curl \
    # For git dependencies (if any)
    git \
    # Clean up
    && rm -rf /var/lib/apt/lists/* \
    && apt-get clean
```

```dockerfile
# ==========================================================
# Builder stage - Install Python dependencies
# ==========================================================
FROM base as builder

# Create virtual environment
RUN python -m venv /opt/venv
ENV PATH="/opt/venv/bin:$PATH"

# Copy requirements first for better caching
COPY docker/requirements/production.txt /tmp/requirements.txt

# Install Python dependencies
RUN pip install --upgrade pip setuptools wheel && \
    pip install -r /tmp/requirements.txt

# ==========================================================
# Production stage - Final image
# ==========================================================
FROM base as production

# Create non-root user for security
RUN groupadd --gid 1000 appgroup && \
    useradd --uid 1000 --gid appgroup --shell /bin/bash --create-home appuser

# Copy virtual environment from builder stage
COPY --from=builder /opt/venv /opt/venv
ENV PATH="/opt/venv/bin:$PATH"
```

```
# Set work directory
WORKDIR /app

# Copy application code
COPY --chown=appuser:appgroup . .

# Remove unnecessary files for production
RUN rm -rf tests/ \
    docker/requirements/development.txt \
    requirements-dev.txt \
    .git/ \
    .pytest_cache/ \
    __pycache__/ \
    *.pyc \
    .coverage

# Create directories for logs and temporary files
RUN mkdir -p /app/logs /app/tmp && \
    chown -R appuser:appgroup /app

# Switch to non-root user
USER appuser

# Health check
HEALTHCHECK --interval=30s --timeout=10s --start-period=30s --retries=3 \
    CMD curl -f http://localhost:5000/health || exit 1

# Expose port
EXPOSE 5000
```

```
# Default command - use gunicorn for production
CMD ["gunicorn", \
    "--bind", "0.0.0.0:5000", \
    "--workers", "4", \
    "--worker-class", "gevent", \
    "--worker-connections", "1000", \
    "--timeout", "120", \
    "--keepalive", "5", \
    "--max-requests", "1000", \
    "--max-requests-jitter", "100", \
    "--preload", \
    "--access-logfile", "-", \
    "--error-logfile", "-", \
    "--log-level", "info", \
    "app:app"]
```

Celery Worker Dockerfile

dockerfile

```dockerfile
# docker/Dockerfile.celery-worker
# Production-optimized Celery worker container

FROM python:3.11-slim as base

ENV PYTHONUNBUFFERED=1 \
    PYTHONDONTWRITEBYTECODE=1 \
    PIP_NO_CACHE_DIR=1 \
    PIP_DISABLE_PIP_VERSION_CHECK=1 \
    PIP_DEFAULT_TIMEOUT=100 \
    C_FORCE_ROOT=1

# Install system dependencies
RUN apt-get update && apt-get install -y --no-install-recommends \
    gcc \
    g++ \
    libpq-dev \
    curl \
    && rm -rf /var/lib/apt/lists/* \
    && apt-get clean

# ==========================================================
# Builder stage
# ==========================================================
FROM base as builder

RUN python -m venv /opt/venv
ENV PATH="/opt/venv/bin:$PATH"
```

```
COPY docker/requirements/production.txt /tmp/requirements.txt
RUN pip install --upgrade pip setuptools wheel && \
    pip install -r /tmp/requirements.txt

# ==========================================================
# Production stage
# ==========================================================
FROM base as production

# Create non-root user
RUN groupadd --gid 1000 celery && \
    useradd --uid 1000 --gid celery --shell /bin/bash --create-home celery

# Copy virtual environment
COPY --from=builder /opt/venv /opt/venv
ENV PATH="/opt/venv/bin:$PATH"

WORKDIR /app

# Copy application code
COPY --chown=celery:celery . .

# Clean up unnecessary files
RUN rm -rf tests/ \
    docker/requirements/development.txt \
    requirements-dev.txt \
    .git/ \
    .pytest_cache/ \
```

```
    __pycache__/ \
    *.pyc \
    .coverage

# Create log directory
RUN mkdir -p /app/logs && chown -R celery:celery /app

USER celery

# Health check for Celery worker
HEALTHCHECK --interval=60s --timeout=30s --start-period=60s --retries=3 \
    CMD celery -A app.celery inspect ping -d celery@$HOSTNAME || exit 1

# Default command
CMD ["celery", \
    "-A", "app.celery", \
    "worker", \
    "--loglevel=info", \
    "--concurrency=4", \
    "--prefetch-multiplier=1", \
    "--max-tasks-per-child=1000", \
    "--time-limit=7200", \
    "--soft-time-limit=3600"]
```

Celery Beat Dockerfile

dockerfile

```
# docker/Dockerfile.celery-beat
# Production-optimized Celery beat scheduler container

FROM python:3.11-slim as base

ENV PYTHONUNBUFFERED=1 \
    PYTHONDONTWRITEBYTECODE=1 \
    PIP_NO_CACHE_DIR=1 \
    PIP_DISABLE_PIP_VERSION_CHECK=1 \
    PIP_DEFAULT_TIMEOUT=100 \
    C_FORCE_ROOT=1

# Install system dependencies
RUN apt-get update && apt-get install -y --no-install-recommends \
    gcc \
    g++ \
    libpq-dev \
    && rm -rf /var/lib/apt/lists/* \
    && apt-get clean

# ==============================================================
# Builder stage
# ==============================================================
FROM base as builder

RUN python -m venv /opt/venv
ENV PATH="/opt/venv/bin:$PATH"
```

```
COPY docker/requirements/production.txt /tmp/requirements.txt
RUN pip install --upgrade pip setuptools wheel && \
    pip install -r /tmp/requirements.txt

# ==================================================================
# Production stage
# ==================================================================
FROM base as production

# Create non-root user
RUN groupadd --gid 1000 celery && \
    useradd --uid 1000 --gid celery --shell /bin/bash --create-home celery

# Copy virtual environment
COPY --from=builder /opt/venv /opt/venv
ENV PATH="/opt/venv/bin:$PATH"

WORKDIR /app

# Copy application code
COPY --chown=celery:celery . .

# Clean up unnecessary files
RUN rm -rf tests/ \
    docker/requirements/development.txt \
    requirements-dev.txt \
    .git/ \
    .pytest_cache/ \
    __pycache__/ \
```

```
    *.pyc \
    .coverage

# Create directories
RUN mkdir -p /app/logs /app/celerybeat-schedule && \
    chown -R celery:celery /app

USER celery

# Health check - check if beat process is running
HEALTHCHECK --interval=60s --timeout=30s --start-period=60s --retries=3 \
    CMD pgrep -f "celery.*beat" || exit 1

# Default command
CMD ["celery", \
    "-A", "app.celery", \
    "beat", \
    "--loglevel=info", \
    "--schedule=/app/celerybeat-schedule/celerybeat-schedule", \
    "--pidfile=/app/celerybeat-schedule/celerybeat.pid"]
```

Requirements Files

Base Requirements (`docker/requirements/base.txt`)

txt

```
# Core Flask dependencies
Flask==2.3.3
Flask-SQLAlchemy==3.0.5
Flask-Login==0.6.3
Flask-WTF==1.1.1
Flask-Migrate==4.0.5
Flask-Limiter==3.5.0
Flask-CORS==4.0.0
Flask-Caching==2.1.0

# Database
SQLAlchemy==2.0.21
psycopg2-binary==2.9.7
alembic==1.12.0

# Celery and Redis
celery==5.3.2
redis==5.0.1

# Security
cryptography==41.0.4
bcrypt==4.0.1

# HTTP and API
requests==2.31.0
urllib3==2.0.5

# Data processing
```

```
pandas==2.1.1
numpy==1.25.2

# Validation
marshmallow==3.20.1
WTForms==3.0.1

# AWS SDK
boto3==1.28.57
botocore==1.31.57

# Utilities
python-dotenv==1.0.0
click==8.1.7
```

Production Requirements (`docker/requirements/production.txt`)

txt

```
-r base.txt

# Production WSGI server
gunicorn[gevent]==21.2.0
gevent==23.7.0

# Production monitoring
sentry-sdk[flask]==1.32.0

# Performance
greenlet==2.0.2

# Email
flask-mail==0.9.1
```

Development Requirements

(`docker/requirements/development.txt`)

```txt
-r production.txt

# Testing
pytest==7.4.2
pytest-cov==4.1.0
pytest-flask==1.2.0
pytest-mock==3.11.1

# Code quality
black==23.9.1
flake8==6.1.0
isort==5.12.0
mypy==1.5.1

# Security testing
bandit==1.7.5
safety==2.3.4

# Development tools
ipython==8.15.0
flask-debugtoolbar==0.13.1
```

Build Scripts

Build Script (`scripts/build-images.sh`)

bash

```bash
#!/bin/bash
# scripts/build-images.sh
# Build all Docker images for ThreatCompass

set -e

# Configuration
REGISTRY=${1:-"local"}
TAG=${2:-"latest"}
REGION=${3:-"us-east-1"}

echo "Building ThreatCompass Docker images..."
echo "Registry: $REGISTRY"
echo "Tag: $TAG"

# Get Git commit SHA for tagging
GIT_SHA=$(git rev-parse --short HEAD)

# Build Flask App
echo "Building Flask App..."
docker build \
    -f docker/Dockerfile.flask-app \
    -t threatcompass-flask-app:$TAG \
    -t threatcompass-flask-app:$GIT_SHA \
    --build-arg BUILD_DATE=$(date -u +'%Y-%m-%dT%H:%M:%SZ') \
    --build-arg GIT_SHA=$GIT_SHA \
    .
```

```
# Build Celery Worker
echo "Building Celery Worker..."
docker build \
    -f docker/Dockerfile.celery-worker \
    -t threatcompass-celery-worker:$TAG \
    -t threatcompass-celery-worker:$GIT_SHA \
    --build-arg BUILD_DATE=$(date -u +'%Y-%m-%dT%H:%M:%SZ') \
    --build-arg GIT_SHA=$GIT_SHA \
    .

# Build Celery Beat
echo "Building Celery Beat..."
docker build \
    -f docker/Dockerfile.celery-beat \
    -t threatcompass-celery-beat:$TAG \
    -t threatcompass-celery-beat:$GIT_SHA \
    --build-arg BUILD_DATE=$(date -u +'%Y-%m-%dT%H:%M:%SZ') \
    --build-arg GIT_SHA=$GIT_SHA \
    .

echo "Build completed successfully!"

# Optional: Push to ECR
if [ "$REGISTRY" != "local" ]; then
    echo "Pushing images to ECR..."

    # Login to ECR
    aws ecr get-login-password --region $REGION | \
        docker login --username AWS --password-stdin $REGISTRY
```

```
    # Tag and push images
    for service in flask-app celery-worker celery-beat; do
        docker tag threatcompass-$service:$TAG $REGISTRY/threatcompass-produc
        docker tag threatcompass-$service:$GIT_SHA $REGISTRY/threatcompass-pr

        docker push $REGISTRY/threatcompass-production/$service:$TAG
        docker push $REGISTRY/threatcompass-production/$service:$GIT_SHA
    done

    echo "Images pushed to ECR successfully!"
fi
```

Development Build Script (`scripts/build-dev.sh`)

```bash
#!/bin/bash
# scripts/build-dev.sh
# Build development images with development dependencies

set -e

echo "Building ThreatCompass development images..."

# Build development Flask app with hot reload
docker build \
    -f docker/Dockerfile.flask-app \
    --target base \
    -t threatcompass-flask-app:dev \
    --build-arg REQUIREMENTS_FILE=docker/requirements/development.txt \
    .

echo "Development build completed!"
```

Docker Compose for Local Development

yaml

```yaml
# docker-compose.dev.yml
version: '3.8'

services:
  web:
    build:
      context: .
      dockerfile: docker/Dockerfile.flask-app
      target: base
    volumes:
      - .:/app
      - /app/docker/requirements/
    ports:
      - "5000:5000"
    environment:
      - FLASK_ENV=development
      - FLASK_DEBUG=1
      - DATABASE_URL=postgresql://postgres:postgres@db:5432/threatcompass_d
      - REDIS_URL=redis://redis:6379/0
    depends_on:
      - db
      - redis
    command: ["flask", "run", "--host=0.0.0.0", "--port=5000", "--reload"]

  celery_worker:
    build:
      context: .
      dockerfile: docker/Dockerfile.celery-worker
```

```
    target: base
  volumes:
    - .:/app
  environment:
    - DATABASE_URL=postgresql://postgres:postgres@db:5432/threatcompass_d
    - REDIS_URL=redis://redis:6379/0
  depends_on:
    - db
    - redis
  command: ["celery", "-A", "app.celery", "worker", "--loglevel=debug", "--reload"]

celery_beat:
  build:
    context: .
    dockerfile: docker/Dockerfile.celery-beat
    target: base
  volumes:
    - .:/app
  environment:
    - DATABASE_URL=postgresql://postgres:postgres@db:5432/threatcompass_d
    - REDIS_URL=redis://redis:6379/0
  depends_on:
    - db
    - redis
  command: ["celery", "-A", "app.celery", "beat", "--loglevel=debug"]

db:
  image: postgres:15
  environment:
```

```yaml
      - POSTGRES_DB=threatcompass_dev
      - POSTGRES_USER=postgres
      - POSTGRES_PASSWORD=postgres
    ports:
      - "5432:5432"
    volumes:
      - postgres_data:/var/lib/postgresql/data

  redis:
    image: redis:7-alpine
    ports:
      - "6379:6379"

volumes:
  postgres_data:
```

Security Optimizations

1. Distroless Images (Advanced)

dockerfile

```dockerfile
# Alternative base using distroless for maximum security
FROM gcr.io/distroless/python3-debian11

COPY --from=builder /opt/venv /opt/venv
COPY --chown=nonroot:nonroot . /app

WORKDIR /app
USER nonroot

ENV PATH="/opt/venv/bin:$PATH"
CMD ["/opt/venv/bin/gunicorn", "app:app"]
```

2. Multi-stage Builds with Vulnerability Scanning

dockerfile

```dockerfile
# Add vulnerability scanning labels
LABEL org.opencontainers.image.source="https://github.com/your-org/threatcomp
LABEL org.opencontainers.image.description="ThreatCompass Flask Application"
LABEL org.opencontainers.image.licenses="MIT"
```

These production-optimized Dockerfiles provide:

✅ **Multi-stage builds** for smaller final images
✅ **Non-root users** for security
✅ **Health checks** for container orchestration

✅ **Optimized caching** with proper layer ordering
✅ **Clean environments** with unnecessary files removed
✅ **Production-ready configurations** for Gunicorn and Celery

The CI/CD pipeline will use these Dockerfiles to build secure, efficient containers for your AWS ECS deployment.