CI/CD Helper Scripts and Test Configuration

These additional files support the comprehensive CI/CD pipeline for ThreatCompass.

Test Configuration

pytest Configuration (`pytest.ini`)

ini

```ini
[tool:pytest]
testpaths = tests
python_files = test_*.py
python_classes = Test*
python_functions = test_*
addopts =
    --strict-markers
    --strict-config
    --verbose
    --tb=short
    --cov=app
    --cov-report=term-missing
    --cov-report=html:htmlcov
    --cov-report=xml:coverage.xml
    --cov-fail-under=80
    --no-cov-on-fail
markers =
    slow: marks tests as slow (deselect with '-m "not slow"')
    integration: marks tests as integration tests
    unit: marks tests as unit tests
    api: marks tests as API tests
    auth: marks tests as authentication tests
    database: marks tests as database tests
filterwarnings =
    ignore::DeprecationWarning
    ignore::PendingDeprecationWarning
```

Test Requirements (`requirements-test.txt`)

```txt
# Testing framework
pytest==7.4.2
pytest-cov==4.1.0
pytest-flask==1.2.0
pytest-mock==3.11.1
pytest-xdist==3.3.1
pytest-env==1.0.1

# Test database
pytest-postgresql==5.0.0

# HTTP testing
responses==0.23.3
httpx==0.25.0

# Factory for test data
factory-boy==3.3.0
faker==19.6.2

# Mocking
freezegun==1.2.2

# Performance testing
pytest-benchmark==4.0.0
```

Test Configuration File (`tests/conftest.py`)

python

```python
# tests/conftest.py
import pytest
import tempfile
import os
from flask import Flask
from app import create_app, db as _db
from db_manager import User, Tenant, Role
from sqlalchemy import event
from sqlalchemy.engine import Engine
import sqlite3


@pytest.fixture(scope='session')
def app():
    """Create application for the tests."""
    # Create a temporary file for the test database
    db_fd, db_path = tempfile.mkstemp()

    app = create_app(test_config={
        'TESTING': True,
        'SQLALCHEMY_DATABASE_URI': f'sqlite:///{db_path}',
        'SQLALCHEMY_TRACK_MODIFICATIONS': False,
        'SECRET_KEY': 'test-secret-key-not-for-production',
        'WTF_CSRF_ENABLED': False,
        'CELERY_TASK_ALWAYS_EAGER': True,
        'CELERY_TASK_EAGER_PROPAGATES': True,
        'REDIS_URL': 'redis://localhost:6379/15',  # Use different DB for tests
        'VT_API_KEY': 'test-vt-api-key',
        'ABUSEIPDB_API_KEY': 'test-abuseipdb-api-key',
    })

    with app.app_context():
        yield app

    # Clean up
    os.close(db_fd)
    os.unlink(db_path)


@pytest.fixture(scope='session')
def db(app):
    """Create database for the tests."""
    _db.app = app

    # Enable foreign key constraints for SQLite
    @event.listens_for(Engine, "connect")
    def set_sqlite_pragma(dbapi_connection, connection_record):
        if isinstance(dbapi_connection, sqlite3.Connection):
            cursor = dbapi_connection.cursor()
            cursor.execute("PRAGMA foreign_keys=ON")
            cursor.close()

    with app.app_context():
        _db.create_all()

        # Create test tenant and roles
        test_tenant = Tenant(name="Test Tenant", subdomain="test")
        _db.session.add(test_tenant)

        admin_role = Role(name="admin", description="Administrator")
        user_role = Role(name="user", description="Regular User")
        _db.session.add(admin_role)
        _db.session.add(user_role)

        _db.session.commit()

        yield _db

        _db.drop_all()


@pytest.fixture(scope='function')
def session(db):
    """Create a clean database session for each test."""
    connection = db.engine.connect()
    transaction = connection.begin()

    options = dict(bind=connection, binds={})
    session = db.create_scoped_session(options=options)

    db.session = session

    yield session

    transaction.rollback()
    connection.close()
    session.remove()



@pytest.fixture
def client(app):
    """Create a test client for the Flask application."""
    return app.test_client()



@pytest.fixture
def runner(app):
    """Create a test runner for the Flask application's Click commands."""
    return app.test_cli_runner()



@pytest.fixture
def auth_headers():
    """Provide authorization headers for API tests."""
    return {'X-API-Key': 'test-api-key'}



@pytest.fixture
def test_user(session):
    """Create a test user."""
    from db_manager import create_user

    user = create_user(
        email="test@example.com",
        username="testuser",
        password="hashedpassword123",
        tenant_id=1
    )
    session.add(user)
    session.commit()
    return user


@pytest.fixture
def admin_user(session):
    """Create an admin test user."""
    from db_manager import create_user

    user = create_user(
        email="admin@example.com",
        username="admin",
        password="hashedpassword123",
        tenant_id=1,
        role_id=1  # Admin role
    )
    session.add(user)
    session.commit()
    return user


@pytest.fixture
def sample_ioc_data():
    """Provide sample IOC data for tests."""
    return {
        'value': '8.8.8.8',
        'type': 'IP_ADDRESS',
        'source': 'test',
        'description': 'Test IOC'
    }


@pytest.fixture
def sample_environment_data():
    """Provide sample environment data for tests."""
    return {
        'tool_type': 'Firewall',
        'tool_name': 'Palo Alto',
        'details': 'Test firewall configuration'
    }


# Mock external services
@pytest.fixture
def mock_virustotal_api(monkeypatch):
    """Mock VirusTotal API responses."""
    def mock_get(*args, **kwargs):
        class MockResponse:
            status_code = 200
            def json(self):
                return {
                    "data": {
                        "attributes": {
                            "last_analysis_stats": {
```

```
                        "malicious": 5,
                        "harmless": 70,
                        "undetected": 10,
                        "suspicious": 2
                    }
                }
            }
        }
    return MockResponse()

monkeypatch.setattr("requests.get", mock_get)


@pytest.fixture
def mock_abuseipdb_api(monkeypatch):
    """Mock AbuseIPDB API responses."""
    def mock_get(*args, **kwargs):
        class MockResponse:
            status_code = 200
            def json(self):
                return {
                    "data": {
                        "abuseConfidenceScore": 85,
                        "countryCode": "US",
                        "totalReports": 10
                    }
                }
        return MockResponse()
    monkeypatch.setattr("requests.get", mock_get)
```

Sample Test Files

API Tests (`tests/test_api.py`)

python

```python
# tests/test_api.py
import pytest
import json
from flask import url_for


class TestIOCAPI:
    """Test IOC API endpoints."""

    def test_create_ioc_success(self, client, auth_headers, sample_ioc_data):
        """Test successful IOC creation."""
        response = client.post(
            '/api/v1/iocs',
            data=json.dumps(sample_ioc_data),
            content_type='application/json',
            headers=auth_headers
        )

        assert response.status_code == 201
        data = json.loads(response.data)
        assert data['status'] == 'success'
        assert 'id' in data['data']

    def test_create_ioc_invalid_data(self, client, auth_headers):
        """Test IOC creation with invalid data."""
        invalid_data = {
            'value': '8.8.8.8',
            'type': 'INVALID_TYPE',  # Invalid IOC type
            'source': 'test'
        }

        response = client.post(
            '/api/v1/iocs',
            data=json.dumps(invalid_data),
            content_type='application/json',
            headers=auth_headers
        )

        assert response.status_code == 400
        data = json.loads(response.data)
        assert data['status'] == 'error'
        assert 'errors' in data

    def test_create_ioc_unauthorized(self, client, sample_ioc_data):
        """Test IOC creation without authentication."""
        response = client.post(
            '/api/v1/iocs',
            data=json.dumps(sample_ioc_data),
            content_type='application/json'
        )

        assert response.status_code == 401


class TestEnvironmentAPI:
    """Test Environment API endpoints."""
```

```
def test_create_environment(self, client, auth_headers, sample_environment_dat
    """Test successful environment creation."""
    response = client.post(
        '/api/v1/environment',
        data=json.dumps(sample_environment_data),
        content_type='application/json',
        headers=auth_headers
    )

    assert response.status_code == 201
    data = json.loads(response.data)
    assert data['status'] == 'success'
    assert 'id' in data['data']
```

Integration Tests (`tests/test_integration.py`)

python

```
# tests/test_integration.py
import pytest
from unittest.mock import patch


@pytest.mark.integration
class TestIOCEnrichmentWorkflow:
    """Test the complete IOC enrichment workflow."""

    def test_ioc_creation_triggers_enrichment(self, client, auth_headers, sample_ioc
                                    mock_virustotal_api, mock_abuseipdb_api):
        """Test that creating an IOC triggers enrichment tasks."""

        with patch('enricher.enrich_ioc_task.delay') as mock_task:
            # Create IOC
            response = client.post(
                '/api/v1/iocs',
                data=json.dumps(sample_ioc_data),
                content_type='application/json',
                headers=auth_headers
            )

            assert response.status_code == 201

            # Verify enrichment task was called
            mock_task.assert_called_once()

    @pytest.mark.slow
```

```python
    def test_playbook_generation_workflow(self, client, auth_headers, sample_ioc_c
        """Test the complete playbook generation workflow."""
        # This would test the full workflow from IOC creation to playbook generation
        pass


@pytest.mark.integration
class TestDatabaseMigrations:
    """Test database migrations and schema changes."""

    def test_database_schema_is_current(self, app):
        """Verify that all migrations are applied."""
        from flask_migrate import current, heads

        with app.app_context():
            current_rev = current()
            head_rev = heads()

            assert current_rev == head_rev, "Database schema is not up to date"
```

Pre-commit Hooks Configuration

Pre-commit Config (`.pre-commit-config.yaml`)

yaml

```yaml
# .pre-commit-config.yaml
repos:
 - repo: https://github.com/pre-commit/pre-commit-hooks
   rev: v4.4.0
   hooks:
     - id: trailing-whitespace
     - id: end-of-file-fixer
     - id: check-yaml
     - id: check-added-large-files
     - id: check-merge-conflict
     - id: debug-statements
     - id: check-json
     - id: pretty-format-json
       args: ['--autofix', '--no-sort-keys']

 - repo: https://github.com/psf/black
   rev: 23.9.1
   hooks:
     - id: black
       language_version: python3.11

 - repo: https://github.com/pycqa/isort
   rev: 5.12.0
   hooks:
     - id: isort
       args: ["--profile", "black"]

 - repo: https://github.com/pycqa/flake8
```

```yaml
    rev: 6.1.0
    hooks:
      - id: flake8
        args: ["--max-line-length=127", "--extend-ignore=E203,W503"]

  - repo: https://github.com/PyCQA/bandit
    rev: 1.7.5
    hooks:
      - id: bandit
        args: ["-r", ".", "-x", "tests/"]

  - repo: https://github.com/python-poetry/poetry
    rev: 1.6.1
    hooks:
      - id: poetry-check

  - repo: local
    hooks:
      - id: pytest-check
        name: pytest-check
        entry: pytest
        language: system
        pass_filenames: false
        always_run: true
        args: ["--maxfail=1", "-x", "-v", "tests/"]
```

GitHub Actions Helper Scripts

ECS Deployment Helper (`scripts/deploy-ecs.sh`)

bash

```bash
#!/bin/bash
# scripts/deploy-ecs.sh
# Helper script for ECS deployments

set -e

# Configuration
CLUSTER_NAME=${1:-"threatcompass-production-cluster"}
SERVICE_NAME=${2}
IMAGE_URI=${3}
REGION=${4:-"us-east-1"}

if [ -z "$SERVICE_NAME" ] || [ -z "$IMAGE_URI" ]; then
   echo "Usage: $0 <cluster_name> <service_name> <image_uri> [region]"
   echo "Example: $0 threatcompass-production-cluster threatcompass-productio
   exit 1
fi

echo "Deploying $SERVICE_NAME with image $IMAGE_URI"

# Get current task definition
TASK_DEFINITION=$(aws ecs describe-task-definition \
   --task-definition $SERVICE_NAME \
   --region $REGION \
   --query 'taskDefinition' \
   --output json)

# Update image URI in task definition
```

```
NEW_TASK_DEFINITION=$(echo $TASK_DEFINITION | jq --arg IMAGE "$IMAGE_
  .containerDefinitions[0].image = $IMAGE |
  del(.taskDefinitionArn, .revision, .status, .requiresAttributes, .placementConstraint

# Register new task definition
echo "Registering new task definition..."
NEW_TASK_DEF_ARN=$(echo $NEW_TASK_DEFINITION | aws ecs register-task-
  --region $REGION \
  --cli-input-json file:///dev/stdin \
  --query 'taskDefinition.taskDefinitionArn' \
  --output text)

echo "New task definition: $NEW_TASK_DEF_ARN"

# Update service with new task definition
echo "Updating ECS service..."
aws ecs update-service \
  --cluster $CLUSTER_NAME \
  --service $SERVICE_NAME \
  --task-definition $NEW_TASK_DEF_ARN \
  --force-new-deployment \
  --region $REGION

# Wait for deployment to complete
echo "Waiting for service to reach stable state..."
aws ecs wait services-stable \
  --cluster $CLUSTER_NAME \
  --services $SERVICE_NAME \
  --region $REGION
```

```
echo "Deployment completed successfully!"

# Get service status
RUNNING_COUNT=$(aws ecs describe-services \
  --cluster $CLUSTER_NAME \
  --services $SERVICE_NAME \
  --region $REGION \
  --query 'services[0].runningCount' \
  --output text)

DESIRED_COUNT=$(aws ecs describe-services \
  --cluster $CLUSTER_NAME \
  --services $SERVICE_NAME \
  --region $REGION \
  --query 'services[0].desiredCount' \
  --output text)

echo "Service Status: $RUNNING_COUNT/$DESIRED_COUNT tasks running"
```

Database Migration Script (`scripts/run-migrations.sh`)

bash

```bash
#!/bin/bash
# scripts/run-migrations.sh
# Run database migrations on ECS

set -e

CLUSTER_NAME=${1:-"threatcompass-production-cluster"}
TASK_DEFINITION=${2:-"threatcompass-production-flask-app"}
REGION=${3:-"us-east-1"}

echo "Running database migrations..."

# Get subnet and security group IDs
SUBNET_ID=$(aws ec2 describe-subnets \
    --filters 'Name=tag:Name,Values=threatcompass-production-private-subnet-1' \
    --region $REGION \
    --query 'Subnets[0].SubnetId' \
    --output text)

SECURITY_GROUP_ID=$(aws ec2 describe-security-groups \
    --filters 'Name=tag:Name,Values=threatcompass-production-ecs-tasks-sg' \
    --region $REGION \
    --query 'SecurityGroups[0].GroupId' \
    --output text)

# Run migration task
TASK_ARN=$(aws ecs run-task \
    --cluster $CLUSTER_NAME \
    --task-definition $TASK_DEFINITION \
    --launch-type FARGATE \
    --region $REGION \
    --network-configuration "awsvpcConfiguration={subnets=[$SUBNET_ID],securi
    --overrides '{
      "containerOverrides": [
        {
          "name": "flask-app",
          "command": ["flask", "db", "upgrade"]
        }
      ]
    }' \
    --tags key=Purpose,value=DatabaseMigration \
    --query 'tasks[0].taskArn' \
    --output text)

echo "Migration task started: $TASK_ARN"

# Wait for task to complete
echo "Waiting for migration to complete..."
aws ecs wait tasks-stopped \
    --cluster $CLUSTER_NAME \
    --tasks $TASK_ARN \
    --region $REGION

# Check task exit code
EXIT_CODE=$(aws ecs describe-tasks \
    --cluster $CLUSTER_NAME \
    --tasks $TASK_ARN \
    --region $REGION \
    --query 'tasks[0].containers[0].exitCode' \
    --output text)

  if [ "$EXIT_CODE" = "0" ]; then
    echo "Database migration completed successfully!"
  else
    echo "Database migration failed with exit code: $EXIT_CODE"
    exit 1
  fi
```

Health Check Script (`scripts/health-check.sh`)

bash

```
#!/bin/bash
# scripts/health-check.sh
# Comprehensive health check for deployed application

set -e

DOMAIN=${1:-$(aws elbv2 describe-load-balancers \
  --names threatcompass-production-alb \
  --query 'LoadBalancers[0].DNSName' \
  --output text)}

REGION=${2:-"us-east-1"}
MAX_RETRIES=${3:-30}
RETRY_INTERVAL=${4:-10}

echo "Running health checks for ThreatCompass deployment..."
echo "Domain: $DOMAIN"

# Function to check endpoint
check_endpoint() {
  local url=$1
  local expected_status=${2:-200}
  local retry_count=0

  echo "Checking $url..."

  while [ $retry_count -lt $MAX_RETRIES ]; do
    if curl -s -o /dev/null -w "%{http_code}" --max-time 10 "$url" | grep -q "^$expe
```

```
            echo "✅ $url is healthy"
            return 0
        fi

        retry_count=$((retry_count + 1))
        echo "❌ Attempt $retry_count/$MAX_RETRIES failed, retrying in ${RETRY_IN
        sleep $RETRY_INTERVAL
    done

    echo "❌ $url failed health check after $MAX_RETRIES attempts"
    return 1
}

# Check main application health
check_endpoint "https://$DOMAIN/health"

# Check API endpoint
check_endpoint "https://$DOMAIN/api/v1/health" 401  # Expect 401 without auth

# Check ECS service health
echo "Checking ECS service health..."
for service in threatcompass-production-flask-app threatcompass-production-cele
    RUNNING_COUNT=$(aws ecs describe-services \
        --cluster threatcompass-production-cluster \
        --services $service \
        --region $REGION \
        --query 'services[0].runningCount' \
        --output text)
```

```
  DESIRED_COUNT=$(aws ecs describe-services \
    --cluster threatcompass-production-cluster \
    --services $service \
    --region $REGION \
    --query 'services[0].desiredCount' \
    --output text)

  if [ "$RUNNING_COUNT" = "$DESIRED_COUNT" ] && [ "$RUNNING_COUNT"
    echo "✅ $service: $RUNNING_COUNT/$DESIRED_COUNT tasks running"
  else
    echo "❌ $service: $RUNNING_COUNT/$DESIRED_COUNT tasks running"
    exit 1
  fi
done

# Check database connectivity
echo "Checking database connectivity..."
DB_STATUS=$(aws rds describe-db-instances \
  --db-instance-identifier threatcompass-production-postgresql \
  --region $REGION \
  --query 'DBInstances[0].DBInstanceStatus' \
  --output text)

if [ "$DB_STATUS" = "available" ]; then
  echo "✅ Database is available"
else
  echo "❌ Database status: $DB_STATUS"
  exit 1
fi
```

```
# Check Redis connectivity
echo "Checking Redis connectivity..."
REDIS_STATUS=$(aws elasticache describe-replication-groups \
  --replication-group-id threatcompass-production-redis \
  --region $REGION \
  --query 'ReplicationGroups[0].Status' \
  --output text)

if [ "$REDIS_STATUS" = "available" ]; then
  echo "✅ Redis is available"
else
  echo "❌ Redis status: $REDIS_STATUS"
  exit 1
fi

echo "🎉 All health checks passed! ThreatCompass is healthy and ready."
```

Code Quality Configuration

Black Configuration (`pyproject.toml`)

toml

```
[tool.black]
line-length = 127
target-version = ['py311']
include = '\.pyi?
extend-exclude = '''
/(
  # directories
  \.eggs
  | \.git
  | \.hg
  | \.mypy_cache
  | \.tox
  | \.venv
  | build
  | dist
  | migrations
)/
'''

[tool.isort]
profile = "black"
multi_line_output = 3
line_length = 127
known_first_party = ["app", "db_manager"]

[tool.coverage.run]
source = ["app"]
omit = [
    "*/tests/*",
    "*/venv/*",
    "*/migrations/*",
    "config.py"
]

[tool.coverage.report]
exclude_lines = [
    "pragma: no cover",
    "def __repr__",
    "raise AssertionError",
    "raise NotImplementedError"
]
```

Flake8 Configuration (`.flake8`)

```ini
[flake8]
max-line-length = 127
exclude =
    .git,
    __pycache__,
    .venv,
    migrations,
    build,
    dist
ignore =
    E203,  # whitespace before ':'
    W503,  # line break before binary operator
    E501   # line too long (handled by black)
per-file-ignores =
    __init__.py:F401
    tests/*:S101,S106
max-complexity = 15
```

This comprehensive CI/CD setup provides:

✅ **Automated testing** with pytest and coverage reporting
✅ **Code quality checks** with Black, isort, flake8, and Bandit
✅ **Security scanning** with Trivy and dependency vulnerability checks
✅ **Database migrations** as part of deployment
✅ **Health checks** and deployment verification

✅ **Notification system** for deployment status

✅ **Pre-commit hooks** for local development quality

The pipeline ensures that every code change is thoroughly tested, scanned for security issues, and deployed reliably to your AWS infrastructure.