Standard Operating Procedure: MuleSoft ESB Setup for Healthcare SOA Project

This document provides detailed step-by-step instructions for setting up, running, testing, troubleshooting, and monitoring the MuleSoft ESB environment within Docker for the Healthcare Information Exchange SOA system.

Table of Contents

Prerequisites

Ensure the following tools are installed on your system:

- Docker Desktop (version 20.10.0+)
- Docker Compose (version 1.29.0+)
- Git (version 2.30.0+)
- Java Development Kit (JDK) 8
- Anypoint Studio 6.6+ (for Mule 3.8.0 development)
- Maven 3.6.0+
- cURL or Postman (for API testing)

Project Setup

1. Clone the Repository

```
git clone https://github.com/your-org/healthcare-soa.git
cd healthcare-soa
```

2. Directory Structure

Ensure your project includes the following directory structure for MuleSoft ESB:

```
healthcare-soa/
├── esb/
│   ├── Dockerfile                    # Custom Mule Runtime Dockerfile
│   ├── apps/                         # Mule applications deployed to the
runtime
│   │   └── healthcare-integration-app/  # Main ESB API
│   ├── domains/                      # Shared domain configurations
│   │   └── default/                  # Default domain configuration
│   ├── conf/                         # Runtime configurations
│   └── logs/                         # Log files
├── services/                         # Microservices
│   ├── patient-service/              # Patient management microservice
│   └── appointment-service/          # Appointment management microservice
├── init-scripts/                     # Database initialization scripts
```

```
├── docker-compose.yml                    # Docker Compose configuration
└── docs/                                 # Project documentation
```

Docker Environment Configuration

1. MuleSoft Dockerfile Setup

Create a Dockerfile for the MuleSoft ESB in `esb/Dockerfile`:

```
FROM openjdk:11-jdk-slim

# Set environment variables
ENV MULE_VERSION=3.8.0
ENV MULE_HOME=/opt/mule
ENV MULE_USER=mule
ENV MULE_USER_UID=1000
ENV MULE_USER_GID=1000

# Install required packages
RUN apt-get update && \
    apt-get install -y --no-install-recommends \
        curl \
        unzip \
        dnsutils \
        iputils-ping \
        procps \
        net-tools && \
    rm -rf /var/lib/apt/lists/*

# Create mule user
RUN groupadd -g ${MULE_USER_GID} ${MULE_USER} && \
    useradd -u ${MULE_USER_UID} -g ${MULE_USER_GID} -s /bin/bash -m ${MULE_USER}

# Install Mule Runtime
RUN mkdir -p ${MULE_HOME} && \
    curl -L https://repository-
master.mulesoft.org/nexus/content/repositories/releases/org/mule/distributions/mule
-standalone/${MULE_VERSION}/mule-standalone-${MULE_VERSION}.tar.gz \
    | tar -xz -C ${MULE_HOME} --strip-components=1 && \
    chown -R ${MULE_USER}: ${MULE_HOME}

# Configure directories for applications
RUN mkdir -p ${MULE_HOME}/apps ${MULE_HOME}/domains ${MULE_HOME}/logs && \
    chown -R ${MULE_USER}: ${MULE_HOME}/apps ${MULE_HOME}/domains ${MULE_HOME}/logs

# Volume configuration
VOLUME ${MULE_HOME}/apps
VOLUME ${MULE_HOME}/domains
VOLUME ${MULE_HOME}/logs
```

```
VOLUME ${MULE_HOME}/conf

# Switch to mule user
USER ${MULE_USER}

# Expose default Mule ports
EXPOSE 8081 8082 5000 9000

# Health check
HEALTHCHECK --interval=30s --timeout=10s --retries=3 CMD curl -f
http://localhost:8081/api/health || exit 1

# Set working directory
WORKDIR ${MULE_HOME}

# Start Mule
CMD ["sh", "-c", "${MULE_HOME}/bin/mule"]
```

2. Docker Compose Configuration

Create the `docker-compose.yml` file in the project root:

```
version: '3.8'

services:
  esb:
    image: vromero/mule:3.8.0
    container_name: healthcare-esb
    ports:
      - "8081:8081"  # HTTP
      - "8082:8082"  # HTTPS
      - "5000:5000"  # JMX
    volumes:
      - ./esb/apps/healthcare-integration-app:/opt/mule/apps/healthcare-
integration-app
      - ./esb/domains/default:/opt/mule/domains/default
    environment:
      - MULE_ENV=local
      - JAVA_OPTS=-Xmx2g -XX:MaxMetaspaceSize=512m
    networks:
      - healthcare-network
    healthcheck:
      test: ["CMD", "curl", "-f", "http://localhost:8081/api/health"]
      interval: 30s
      timeout: 10s
      retries: 3

  postgres:
    image: postgres:13
    container_name: healthcare-postgres
    ports:
      - "5432:5432"
```

```
    environment:
      - POSTGRES_USER=postgres
      - POSTGRES_PASSWORD=postgres123456
      - POSTGRES_DB=healthcaredb
    volumes:
      - postgres-data:/var/lib/postgresql/data
      - ./init-scripts:/docker-entrypoint-initdb.d
    networks:
      - healthcare-network

  mongodb:
    image: mongo:5.0
    container_name: healthcare-mongodb
    ports:
      - "27017:27017"
    environment:
      - MONGO_INITDB_ROOT_USERNAME=mongouser
      - MONGO_INITDB_ROOT_PASSWORD=mongopassword
    volumes:
      - mongo-data:/data/db
    networks:
      - healthcare-network

  redis:
    image: redis:6.2
    container_name: healthcare-redis
    ports:
      - "6379:6379"
    command: redis-server --requirepass redispassword
    volumes:
      - redis-data:/data
    networks:
      - healthcare-network

  kafka:
    image: confluentinc/cp-kafka:7.0.0
    container_name: healthcare-kafka
    depends_on:
      - zookeeper
    ports:
      - "9092:9092"
    environment:
      - KAFKA_BROKER_ID=1
      - KAFKA_ZOOKEEPER_CONNECT=zookeeper:2181
      -
KAFKA_ADVERTISED_LISTENERS=PLAINTEXT://kafka:29092,PLAINTEXT_HOST://localhost:9092
      -
KAFKA_LISTENER_SECURITY_PROTOCOL_MAP=PLAINTEXT:PLAINTEXT,PLAINTEXT_HOST:PLAINTEXT
      - KAFKA_INTER_BROKER_LISTENER_NAME=PLAINTEXT
      - KAFKA_OFFSETS_TOPIC_REPLICATION_FACTOR=1
    networks:
      - healthcare-network

  zookeeper:
    image: confluentinc/cp-zookeeper:7.0.0
    container_name: healthcare-zookeeper
    ports:
```

```
      - "2181:2181"
    environment:
      - ZOOKEEPER_CLIENT_PORT=2181
    volumes:
      - zookeeper-data:/var/lib/zookeeper/data
      - zookeeper-log:/var/lib/zookeeper/log
    networks:
      - healthcare-network

networks:
  healthcare-network:
    driver: bridge

volumes:
  postgres-data:
  mongo-data:
  redis-data:
  zookeeper-data:
  zookeeper-log:
```

3. PostgreSQL Initialization Script

Create the database initialization script in `init-scripts/init-postgres.sh`:

```
#!/bin/bash
set -e

# Function to create databases
create_db() {
  local db=$1
  echo "Creating database: $db"
  psql -v ON_ERROR_STOP=1 --username "$POSTGRES_USER" --dbname "$POSTGRES_DB" <<-
EOSQL
    CREATE DATABASE $db;
    GRANT ALL PRIVILEGES ON DATABASE $db TO $POSTGRES_USER;
EOSQL
}

# Create each database
if [ -n "$POSTGRES_MULTIPLE_DATABASES" ]; then
  echo "Creating multiple databases: $POSTGRES_MULTIPLE_DATABASES"
  for db in $(echo $POSTGRES_MULTIPLE_DATABASES | tr ',' ' '); do
    create_db $db
  done
  echo "Multiple databases created"
fi
```

Make sure to set the execution permission:

```
chmod +x init-scripts/init-postgres.sh
```

MuleSoft Application Development

1. Create a Basic MuleSoft Application

Create an integration application in `esb/apps/healthcare-integration-app/` with the following files:

mule-deploy.properties:

```
# Mule deployment descriptor
redeployment.enabled=true
encoding=UTF-8
config.resources=healthcare-integration-app.xml
domain=default
```

healthcare-integration-app.xml:

```
<?xml version="1.0" encoding="UTF-8"?>
<mule xmlns="http://www.mulesoft.org/schema/mule/core"
      xmlns:http="http://www.mulesoft.org/schema/mule/http"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:schemaLocation="
        http://www.mulesoft.org/schema/mule/core
http://www.mulesoft.org/schema/mule/core/current/mule.xsd
        http://www.mulesoft.org/schema/mule/http
http://www.mulesoft.org/schema/mule/http/current/mule-http.xsd">

    <http:listener-config name="HTTP_Listener_config" host="0.0.0.0" port="8081" />
    <http:request-config name="Patient_Service_Request_config" host="patient-
service" port="8091" />
    <http:request-config name="Appointment_Service_Request_config"
host="appointment-service" port="8092" />

    <!-- Health check endpoint -->
    <flow name="health-check-flow">
        <http:listener config-ref="HTTP_Listener_config" path="/api/health" />
        <set-payload value='{"status":"UP","timestamp":"now","components":{"esb":
{"status":"UP"}}}' mimeType="application/json" />
    </flow>

    <!-- Patient service gateway -->
    <flow name="patient-api-flow">
```

```
        <http:listener config-ref="HTTP_Listener_config" path="/api/v1/patients/*"
/>
        <http:request config-ref="Patient_Service_Request_config"
path="/api/patients/{path}" method="#[attributes.method]">
            <http:uri-params>
                <![CDATA[#[output application/java
                ---
                {
                    "path": attributes.uriParams.path
                }]]]>
            </http:uri-params>
        </http:request>
    </flow>

    <!-- Appointment service gateway -->
    <flow name="appointment-api-flow">
        <http:listener config-ref="HTTP_Listener_config"
path="/api/v1/appointments/*" />
        <http:request config-ref="Appointment_Service_Request_config"
path="/api/appointments/{path}" method="#[attributes.method]">
            <http:uri-params>
                <![CDATA[#[output application/java
                ---
                {
                    "path": attributes.uriParams.path
                }]]]>
            </http:uri-params>
        </http:request>
    </flow>
</mule>
```

2. Configuration Files

Create the configuration file for local environment in `esb/apps/healthcare-integration-app/healthcare-config-local.yaml`:

```
# Healthcare SOA Configuration - Local Environment

# Service Hostnames and Ports
patient.service.host: "patient-service"
patient.service.port: 8091
appointment.service.host: "appointment-service"
appointment.service.port: 8092

# Database Configuration
postgres.db.url: "jdbc:postgresql://postgres:5432/healthcaredb"
postgres.db.username: "postgres"
postgres.db.password: "postgres123456"
mongo.db.url: "mongodb://mongouser:mongopassword@mongodb:27017/appointmentdb"
redis.url: "redis://redispassword@redis:6379"
```

```
# Enterprise Database Adaptation Strategy
db.use.legacy.schema: true
db.feature.flags.enabled: true

# Other configurations as needed
```

3. Logging Configuration

Create a logging configuration in `esb/conf/log4j2.xml`:

```
<?xml version="1.0" encoding="UTF-8"?>
<Configuration status="INFO">
    <Appenders>
        <Console name="Console" target="SYSTEM_OUT">
            <PatternLayout pattern="%d{HH:mm:ss.SSS} [%t] %-5level %logger{36} -
%msg%n"/>
        </Console>
        <RollingFile name="RollingFile" fileName="/opt/mule/logs/healthcare-
esb.log"
                     filePattern="/opt/mule/logs/healthcare-esb-%d{yyyy-MM-dd}-
%i.log">
            <PatternLayout pattern="%d{HH:mm:ss.SSS} [%t] %-5level %logger{36} -
%msg%n"/>
            <Policies>
                <SizeBasedTriggeringPolicy size="10 MB"/>
                <TimeBasedTriggeringPolicy interval="1" modulate="true"/>
            </Policies>
            <DefaultRolloverStrategy max="10"/>
        </RollingFile>
    </Appenders>
    <Loggers>
        <Root level="info">
            <AppenderRef ref="Console"/>
            <AppenderRef ref="RollingFile"/>
        </Root>
        <Logger name="com.healthcare" level="DEBUG"/>
        <Logger name="org.mule.runtime" level="INFO"/>
    </Loggers>
</Configuration>
```

Running the Environment

1. Build and Start Docker Containers

Navigate to the project root directory and run:

```
docker-compose build
docker-compose up -d
```

This command builds the Docker images and starts all containers.

2. Verify Container Status

Ensure all containers are running:

```
docker-compose ps
```

The output should show all services (esb, postgres, mongodb, redis, kafka, zookeeper) as "Up" with their respective ports.

3. Check MuleSoft ESB Logs

View the logs from the MuleSoft container:

```
docker-compose logs -f esb
```

Look for the startup confirmation message: "Started app 'healthcare-integration-app'".

4. Access the Health Check Endpoint

Verify the ESB is operational by accessing the health check:

```
curl http://localhost:8081/api/health
```

Expected response:

```
{
  "status": "UP",
  "timestamp": "2025-05-17T23:05:27.123Z",
  "components": {
```

```
    "esb": {
      "status": "UP"
    }
  }
}
```

Testing the ESB

1. Functional Testing

Basic Health Check Test

```
curl -v http://localhost:8081/api/health
```

Verify that:

- HTTP status code is 200
- Response contains `"status": "UP"`

Service Integration Test

Once you've implemented service integration flows:

```
curl http://localhost:8081/api/patients/123
```

Check that the ESB orchestrates calls to the correct services and returns appropriate responses.

2. Load Testing

Use tools like Apache JMeter or k6 to assess performance:

```
# Example k6 command
k6 run --vus 10 --duration 30s test-scripts/health-check-test.js
```

Monitor response times and error rates.

3. Integration Testing

For cross-service functionality, create test cases that verify:

- Proper orchestration of service calls
- Error handling and fallback mechanisms
- Transaction management

Troubleshooting

1. Container Issues

MuleSoft Container Not Starting

If the MuleSoft container fails to start:

```
# Check logs
docker-compose logs esb

# Access container for inspection
docker-compose exec esb /bin/bash

# Verify Mule runtime is installed correctly
ls -la /opt/mule
```

Common issues:

- Insufficient permissions on mounted volumes
- Missing required files
- Port conflicts

Database Connection Issues

If services can't connect to databases:

```
# Verify PostgreSQL is running
docker-compose ps postgres
```

```
# Check database initialization
docker-compose exec postgres psql -U postgres -c "\l"
```

2. MuleSoft Application Issues

Application Not Deploying

If the MuleSoft application fails to deploy:

1. Check the application structure:

```
ls -la esb/apps/healthcare-integration-app/
```

2. Verify the application artifact:

```
docker-compose exec esb ls -la /opt/mule/apps
```

3. Check deployment logs:

```
docker-compose exec esb cat /opt/mule/logs/mule-app-healthcare-integration-app.log
```

Runtime Errors

For errors during execution:

1. Check runtime logs:

```
docker-compose exec esb cat /opt/mule/logs/mule_ee.log
```

2. Enable debug logging by updating `log4j2.xml` to set root level to "DEBUG".

3. Common MuleSoft Errors

Error Message	Possible Cause	Solution
`Failed to deploy artifact`	Invalid application structure	Verify mule-deploy.properties and XML files
`Could not find a transformer`	MEL transformation issues	Check expression syntax
`Connection timeout`	Service not available	Verify service host and port
`OutOfMemoryError`	Insufficient memory allocation	Increase JVM memory settings
`Legacy database schema error`	Incompatibility with enterprise schema	Apply enterprise database adaptation strategy

Monitoring

1. Basic Monitoring

Container Health Checks

Docker Compose includes health checks. Monitor them with:

```
docker-compose ps
```

Look for "healthy" status for each container.

Log Monitoring

Monitor logs in real-time:

```
docker-compose logs -f esb
```

2. Advanced Monitoring

JMX Monitoring

MuleSoft exposes JMX metrics on port 5000. Connect using tools like JConsole:

```
jconsole localhost:5000
```

Prometheus Integration

Integrate with Prometheus for metrics collection:

1. Add the Prometheus dependency to your Mule application.
2. Configure a metrics endpoint in your application.
3. Set up Prometheus to scrape metrics from `http://localhost:8081/metrics`.

Grafana Dashboards

Create Grafana dashboards to visualize:

- JVM metrics (Memory, CPU, Threads)
- Application metrics (request rate, error rate)
- Custom business metrics

3. Alerts and Notifications

Set up alerts for:

- Container status changes
- High CPU/memory usage
- Elevated error rates
- Slow response times

Deployment Pipeline

1. CI/CD Pipeline Setup

Create a CI/CD pipeline using Jenkins, GitHub Actions, or Azure DevOps with the following stages:

1. **Build**: Compile and package Mule applications

2. **Test**: Run automated tests
3. **Docker Build**: Create Docker images
4. **Deploy**: Deploy to target environment

2. Environment Deployment

For each environment (dev, test, staging, production):

1. Create environment-specific configuration files:
    - `healthcare-config-dev.yaml`
    - `healthcare-config-test.yaml`
    - `healthcare-config-prod.yaml`
2. Update the Docker Compose files for each environment with appropriate settings.
3. Set up a deployment script to:
    - Build the required Docker images
    - Deploy to the target environment
    - Verify the deployment

Best Practices

1. MuleSoft Docker Best Practices

- **Image Selection**: Use community-supported images like `vromero/mule:3.8.0`
- **Security**: Remove unnecessary tools from production images
- **Configuration**: Use environment variables for all configurations
- **Logging**: Implement structured logging
- **Health Checks**: Include comprehensive health checks
- **Resource Limits**: Set appropriate CPU and memory limits
- **Volume Management**: Only mount necessary directories

2. MuleSoft Application Best Practices

- **Modularization**: Split functionality into separate Mule applications

- **Error Handling**: Implement comprehensive error handling
- **Caching**: Use caching for frequently accessed data
- **Transactions**: Handle transactions appropriately
- **API Contracts**: Define clear API contracts
- **Documentation**: Document all APIs using RAML
- **Testing**: Implement automated testing
- **Domain Sharing**: Utilize domains for shared resources

3. Security Best Practices

- **Secret Management**: Use Docker secrets or external vault for sensitive data
- **Network Isolation**: Configure appropriate network segregation
- **Image Scanning**: Scan Docker images for vulnerabilities
- **API Security**: Implement OAuth 2.0 or other security protocols
- **Least Privilege**: Run containers with minimal permissions
- **Regular Updates**: Keep all components updated

4. Enterprise Database Adaptation Best Practices

- **Schema Compatibility**: Use `@Column(columnDefinition="serial")` for ID fields
- **Type Consistency**: Align Java types with database column types
- **Transient Properties**: Use `@Transient` for missing columns with custom getters/setters
- **Repository Patterns**: Implement custom repository interfaces for flexibility
- **Feature Flags**: Use flags to control feature availability
- **Hibernate Configuration**: Set `ddl-auto: none` instead of `validate`
- **Naming Strategies**: Configure proper naming strategies for working with legacy schemas

Conclusion

This SOP provides a comprehensive guide for setting up, running, testing, troubleshooting, and monitoring a MuleSoft ESB environment within Docker for the

Healthcare Information Exchange SOA system.

By following these instructions, you can establish a robust integration platform that follows SOA principles and twelve-factor application methodology, enabling effective healthcare information exchange across disparate systems while successfully adapting to enterprise database schemas.