1. IPF Open eHealth Integration Platform - Build IPF Docker Image

1.1 Introduction

In this article detailed steps for creating a Docker image containing the IPF Open eHealth Platform environment based the IPF 3.7-SNAPSHOT are provided. This image can be used for exploration of the IPF platform and its healthcare integration capabilities, as well as generic Apache Camel integration platform features and capabilities that complement the healthcare capabilities of the IPF.

This article could be of interest to these who are aware of healthcare integration standards, health information exchange standards and requirements of healthcare integration. Specifically, these who would like to explore the IPF, a FOSS platform (Free, Open Source Software), to determine whether to deploy it in a healthcare environment as a healthcare integration platform that supports common healthcare interoperability standards.

IPF supports the following features related to the eHealth domain (from https://oehf.github.io/ipf/):

Feature	Description
Support for eHealth integration profiles	A set of components for creating actor interfaces as specified in IHE and Continua integration profiles. IPF currently supports creation of actor interfaces for the IHE profiles XDS.a, XDS.b, PIX, PDQ, PIXv3, PDQv3, PIXm, PDQm, MHD, QED, XCPD, XCA, XCA-I, XCF, XPID, PCD, as well as for Continua profiles HRN and WAN.
HL7 Message processing	Basis for HL7 message processing is the HL7v2 DSL. These provides the basis for implementing HL7 Message processing Camel routes.
HL7 Message translation	Translation utilities for translating between HL7v3 and HL7v2 messages for corresponding IHE transactions
CDA Support	Wrapping a number of CDA-related libraries, providing the basis for implementing CDA processing Camel routes.
FHIR Support	FHIR® – Fast Healthcare Interoperability Resources (hl7.org/fhir) – is a next generation standards framework created by HL7 leveraging the latest web standards and applying a tight focus on implementability.
DICOM Audit Support	Support for constructing, serializing and sending DICOM audit messages

Other IPF features provide part of the underlying foundation or supporting functionality:

Feature	Description
Core Features	Domain-neutral message processors and DSL extensions usable for general-purpose message processing.
Code System Mapping	A simplistic mechanism for mapping code values between code systems
Dynamic Feature Registration	Aids in building up modular integration solutions where each module contributes routes, services etc. to the overall application

The Dockerfile and docker-compose.yml files, that will allow the image to be created, are provided in the article.

At the end of the process a Docker Image which includes the compiled and tested IPF 3.7-SNAPSHOT will be available for creating docker containers ready for healthcare integration work.

This article consists of the following tasks:

1. Create Baseline Docker Image
2. Compile and Test IPF
3. Create Updated Image with IPF Built and Tested
4. Create and Briefly Explore the Work Container

See IPF Open eHealth Integration Platform for details of IPF - https://oehf.github.io/ipf/.

See Docker for details on obtaining, installing, configuring and using Docker infrastructure - https://www.docker.com/.

1.2 Pre-Requisites

This article assumes that the Docker Desktop for Windows or Linux is installed on the physical machine (hereafter Host) so that Docker Images can be created and Docker Containers based on these images can be run (hereafter Containers).

Operating System commands, tools file system paths, etc., used in this article are Linux commands, tools, file system paths, etc.. The few Host commands that are used can be readily converted for the Windows environment but don't have to be is the Host platform is Windows 10 with the Windows Subsystem for Linux (hereafter WSL). This is the environment used in article.

1.3 Create Baseline Docker Image

1.3.1 Establish Host environment

It is assumed that the host is a Linux system, either directly or through the Windows 10 Windows Subsystem for Linux. I used the later for this work, running Debian 10.

Create directory hierarchy for IPF development (replace /mnt/c/ipf, which is a root of my host development environment with your own path)

```
cat <<-EOF >> ~/.bash_profile

export IPF_HOME=/mnt/c/ipf
export DOCKER_REP=${DOCKER_REP}

EOF
source ~/.bash_profile
```

Create the hierarchy under the IPF_HOME root

```
mkdir -pv ${IPF_HOME}/{Downloads,docker,shared,ipfdev}
```

```
mkdir: created directory '/mnt/c/ipf'
mkdir: created directory '/mnt/c/ipf/Downloads'
mkdir: created directory '/mnt/c/ipf/docker'
mkdir: created directory '/mnt/c/ipf/shared'
mkdir: created directory '/mnt/c/ipf/ipfdev'
```

1.3.2 Create Docker Image

Create the Dockerfile which will be used to build the baseline docker image. Note that unzip, wget, tzdata, maven and openjdk-8-jdk are required packages. All other packages are optional and

included for my convenience. unzip is required to unpack the ipf-master.zip. wget is required to get the settings.xml form the remote site. tzdata is required to set the timezone. Once the image is created, these packages can be removed is you don't want to use them. openjdk-8-jdk is required at build time and at runtime and must remain.

Modify values of the ENV IPF_* and TZ_* environment values as required for your environment. These environment variables are used during the image build process and inside the container built from the image. There is no real need to change them, except perhaps TZ_* variables if you want to use a different timezone.

```
cd ${IPF_HOME}
```

```
cat <<-'EOF' > ${IPF_HOME}/docker/Dockerfile.baseline
```

```
FROM debian

ENV IPF_USER=ipf
ENV IPF_UID=1000
ENV IPF_GID=1000
ENV IPF_ROOT=/ipf
ENV IPF_HOME=${IPF_ROOT}/ipf-master
ENV IPF_DEV=${IPF_ROOT}/dev
ENV JAVA_HOME=/usr/lib/jvm/java-8-openjdk-amd64/
ENV TZ_PATH=Australia/Sydney
ENV TZ_NAME=Australia/Sydney

RUN apt-get update && \
    apt-get -y upgrade && \
    apt-get -y install \
      apt-utils && \
    cp -v /etc/apt/sources.list /etc/apt/sources.list_orig && \
    echo deb http://ftp.de.debian.org/debian sid main >> /etc/apt/sources.list && \
    apt-get update && \
    apt-get -y upgrade && \
    apt-get -y install \
      sudo \
      zip \
      tree \
      net-tools \
      iputils-ping \
      curl \
      gnupg2 \
      procps \
      htop \
      iftop \
      httpie \
      dos2unix \
      jq \
      nano \
      unzip \
      wget \
      tzdata \
      maven  \
      openjdk-8-jdk && \
    update-alternatives --install /usr/bin/javac javac ${JAVA_HOME}/bin/javac 0 && \
    update-alternatives --install /usr/bin/java java ${JAVA_HOME}/bin/java 0 && \
    update-alternatives --set javac ${JAVA_HOME}/bin/javac && \
    update-alternatives --set java ${JAVA_HOME}/bin/java && \
    java -version && \
    javac -version && \
    cp -v /usr/share/zoneinfo/${TZ_PATH} /etc/localtime && \
    echo "${TZ_NAME}" >  /etc/timezone && \
    addgroup --gid ${IPF_GID} ipf && \
    useradd -c 'IPF non-root user' -u ${IPF_UID} -g ${IPF_GID} -m -p ipf -s /bin/bash
${IPF_USER} && \
    mkdir -pv ${IPF_ROOT} && \
    chown -Rv ${IPF_UID}:${IPF_GID} ${IPF_ROOT}
```

```
USER ${IPF_UID}:${IPF_GID}
RUN mkdir -pv /home/ipf/Downloads && \
    wget https://codeload.github.com/oehf/ipf/zip/master -O /home/ipf/Downloads/ipf-master.zip
&& \
    unzip /home/ipf/Downloads/ipf-master.zip -d ${IPF_ROOT} && \
    wget -q https://raw.githubusercontent.com/oehf/ipf-labs/master/maven/settings.xml --no-
check-certificate -O ${IPF_HOME}/settings.xml &&\
    echo "MAVEN_OPTS=-Xmx3g" > /home/ipf/.mavenrc && \
    echo "export MAVEN_OPTS=-Xmx3g" >> /home/ipf/.bash_profile && \
    echo "export PATH=${JAVA_HOME}/bin:${PATH}" >> /home/ipf/.bash_profile
```

EOF

Build the docker image

```
cd ${IPF_HOME}/docker
```

```
docker build  --no-cache --rm --tag ipf_base:1.0.0 --file ./Dockerfile.baseline . |
tee ./ipf_base_1.0.0_image_build.log
```

Start the image

```
docker run -d -itu root -w /home/ipf --name ipf_base --rm ipf_base:1.0.0
```

Attach to the running image and look around. When you exit from the shell the image will stop.

```
docker attach ipf_base
```

Save the docker-compose.yml, if any already exists in this location and create a new one.

```
cd ${IPF_HOME}
```

```
if [ ! -f ./docker/docker-compose.yml.1.0.0 ]; then cp -v ./docker/docker-
compose.yml ./docker/docker-compose.yml.1.0.0; fi
```

This docker-compose file defines the network that this, and related containers if any, would use to communicate, and mounts the shared host directory in the container. This shared directory can be used for bi-directional file exchange.

```
cat <<-'EOF' > ./docker/docker-compose.yml
```

```
version: "3.1"

services:
    ipf_base:
        container_name: ipf_base
        image: ipf_base:1.0.0
        # restart: always
        tty: true
        stdin_open: true
        #        expose:
        #            - '9000'
        #        ports:
        #            - 9000:9000
        networks:
            ipf_net:
                aliases:
                    - ipf_base
        hostname: ipf_base
        volumes:
            - /var/run/docker.sock:/var/run/docker.sock
            - ../shared:/shared
#
networks:
    ipf_net:
        driver: bridge
```

EOF

Start the container in detached mode

```
cd ${IPF_HOME}/docker

docker-compose up --detach --remove-orphans ipf_base
```

Connect to the container and explore. When you exit the container will continue to run.

```
docker exec -itu root ipf_base bash -l
```

Strictly speaking there is no need to push the image to a repository. If you don't have one or don't want to bother then ignore commands with ${DOCKER_REP}/.

```
docker tag ipf_base:1.0.0 ${DOCKER_REP}/ipf_base:1.0.0
docker stop ipf_base
docker push ${DOCKER_REP}/ipf_base:1.0.0
```

1.3.3 Compile and Test IPF

By this time the image has all the packages installed and the ipf-master.zip archive extracted to the location pointed to by the IPF_ROOT guest environment variable set in the Dockerfile above.

If you are using a remote repository for your images, then you should test pulling the image from it. If not, skip the following two commands.

```
docker pull ${DOCKER_REP}/ipf_base:1.0.0

docker tag ${DOCKER_REP}/ipf_base:1.0.0 ipf_base:1.0.0
```

Now that the image is available, let's create and start the docker container based on this image with selected Host directories shared with the Container. The following command starts the container and the command after that tests that it is running and the shared directory is accessible from within the container.

```
docker-compose up -d --remove-orphans ipf_base

docker exec -itu root ipf_base ls -al /shared
```

1.3.4 Build IPF

The ipf-master.zip archive has been extracted to the IPF_ROOT directory in the Docker Image. We can now use the Maven framework to build and test IPF.

Maven clean, validate, compile, test, package, integration-test, verify, install targets will download a variety of dependencies first time they are triggered. It will take a while and will require the container to be able to access the Internet. Depending on which of the targets you run, what your Internet connectivity is like and what your host's performance characteristics and resources are like, it may take an hour or more to complete this stage of image creation.

```
WSL

docker exec -itu ipf -w /ipf/ipf-master ipf_base mvn clean --settings settings.xml

docker exec -itu ipf -w /ipf/ipf-master ipf_base mvn compile --settings
settings.xml
```

Note that all components compile without issues. Note, however, that tests for the ipf-commons-audit component will fail because ATNA server is not configured in the environment. Thereafter ipf-commons-audit will be excluded so that the following components can be tested and packaged.

Run the full test to see what the errors look like and what happens when one of the tests fails.

```
docker exec -itu ipf -w /ipf/ipf-master ipf_base mvn test --settings settings.xml #
fails on ipf-commons-audit and aborts
```

Now run one or more of the phases below. The test phase will run all the phases up to and including the test phase - run this to make sure that all tests, except these for the ipf-commons-audit component run to completion. See Maven in 5 Minutes if you are not familiar with Apache Maven to understand this comment. (https://maven.apache.org/guides/getting-started/maven-in-five-minutes.html)

```
docker exec -itu ipf -w /ipf/ipf-master ipf_base mvn test --settings settings.xml -
pl '!:ipf-commons-audit'
```

You can execute subsequent phases one at a time if you are curious. The install phase will execute all phases up to and including install. Package, integration-test and verify phases will be executed implicitly anyway.

```
docker exec -itu ipf -w /ipf/ipf-master ipf_base mvn package --settings
settings.xml -pl '!:ipf-commons-audit'

docker exec -itu ipf -w /ipf/ipf-master ipf_base mvn integration-test --settings
settings.xml -pl '!:ipf-commons-audit'

docker exec -itu ipf -w /ipf/ipf-master ipf_base mvn verify --settings settings.xml
-pl '!:ipf-commons-audit'

docker exec -itu ipf -w /ipf/ipf-master ipf_base mvn install --settings
settings.xml -pl '!:ipf-commons-audit'
```

1.3.5 Create Updated Image with IPF Built and Tested

Create a new baselines image with hundreds of megabytes of dependencies in the local ~/.m2/repository hierarchy, so that each new container based on this image will have them available and will not need to download them from the remote repository all over again.

Save the image checkpoint and stop the image.

```
docker commit ipf_base ipf_base:1.0.1

docker stop ipf_base
```

Ignore the following two commands if you are not using a remote repository.

```
docker tag ipf_base:1.0.1 ${DOCKER_REP}/ipf_base:1.0.1

docker push ${DOCKER_REP}/ipf_base:1.0.1
```

If you are not using remote repository for your images then skip the following command. Optionally remove images from the local environment if you are using one.

```
docker image rm ${DOCKER_REP}/ipf_base:1.0.1 ${DOCKER_REP}/ipf_base:1.0.0
ipf_base:1.0.1 ipf_base:1.0.0
```

1.3.6 Update docker-compose.yml to use the ipf_base:1.0.1 image

Not that the container will mount Host directories ${IPF_HOME}/shared and ${IPF_HOME}/ipfdev in the Container. This means that the content of these directories will be accessible from the host and from the container.

If you want to run the Eclipse IDE as a development IDE on the host, for example, and build applications inside the container then make sure that you create your projects in the ${IPF_HOME}/ipfdev on the host and they will be accessible in the /ipf/dev in the container.

Create a new docker-compose.yml configuration so that future ipf_base containers will use the 1.0.1 image and will mount the two shared directories.

```
cd ${IPF_HOME}

if [ ! -f ./docker/docker-compose.yml.1.0.1 ]; then cp -v ./docker/docker-
compose.yml ./docker/docker-compose.yml.1.0.1; fi

cat <<-'EOF' > ./docker/docker-compose.yml
```

```
version: "3.1"

services:
    ipf_base:
        container_name: ipf_base
        image: ipf_base:1.0.1
        # restart: always
        tty: true
        stdin_open: true
        expose:
            - '9000'
        networks:
            ipf_net:
                aliases:
                    - ipf_base
        hostname: ipf_base
        volumes:
            - /var/run/docker.sock:/var/run/docker.sock
            - ../shared:/shared
            - ../ipfdev:/ipf/dev
#
networks:
    ipf_net:
        driver: bridge
```

```
EOF
```

1.3.7 Create and Briefly Explore the Work Container

Create new container using the ipf_base:1.0.1 image as the base for further work.

```
cd ${IPF_HOME}/docker
```

Skip the following two commands if you are not pulling the image from the remote repository.

```
docker pull ${DOCKER_REP}/ipf_base:1.0.1

docker tag ${DOCKER_REP}/ipf_base:1.0.1 ipf_base:1.0.1
```

Start the new container.

```
docker-compose up -d --remove-orphans ipf_base
```

Test guest-host sharing

```
cd ${IPF_HOME}

docker exec -itu ipf ipf_base ls -al /shared /ipf/dev

docker exec -itu ipf -w /home/ipf ipf_base bash -l -c 'echo "Hi there" >
/shared/aa.xx && echo "Hi there" > /ipf/dev/aa.xx'

docker exec -itu ipf ipf_base ls -al /shared /ipf/dev

ls -al ${IPF_HOME}/shared ${IPF_HOME}/ipfdev

rm -v ${IPF_HOME}/shared/aa.xx ${IPF_HOME}/ipfdev/aa.xx
```

Verify that environment variables are set as expected

```
docker exec -itu ipf ipf_base bash -lc 'set | grep IPF_'
```

The container is ready for work. Log in as ipf and explore. When you are done proceed to tutorials if you are so inclined.

```
docker exec -itu ipf -w /home/ipf ipf_base bash -l
```

1.4 Summary

In this article detailed steps for creating a Docker image containing the IPF Open eHealth Platform environment based the IPF 3.7-SNAPSHOT as at early February 2020 were provided. The resulting Docker image, and Docker Containers created from it, can be used for exploration of the IPF platform and its healthcare integration capabilities, as well as generic Apache Camel integration platform features.

This article could be of interest to these who are aware of healthcare integration standards, health information exchange standards and requirements of healthcare integration. Specifically these who would like to explore the IPF, a FOSS platform (Free, Open Source Software), to determine whether to deploy it in a healthcare as a healthcare integration platform that supports common interoperability healthcare standards.