How to set up a local DMaaP installation in Docker for testing

It is sometimes convenient to have a local installation of DMaaP on your machine for testing purposes. For example, for testing policies, you can connect the Policy Framework to a local installation of DMaaP and use curl to issue the policy REST calls to the local DMaaP. For this howto, in order to avoid a dependency on A&AI, checks towards A&AI are commented out. This howto was written on the Amsterdam versions of DMaaP. It was executed on a Macbook Pro running macOS 10.13.3.

Step-by-step guide

1. Create a new directory, let's call it "local-dmaap", and go into the directory

```
mkdir <path-to>/local-dmaap
cd  <path-to>/local-dmaap
```

2. Clone the DMaaP msgrtr and messageservice repositories

```
git clone https://gerrit.onap.org/r/dmaap/messagerouter/msgrtr
git clone https://gerrit.onap.org/r/dmaap/messagerouter/messageservice
```

3. Check out the Amsterdam branches of the DMaaP repos

```
cd msgrtr
git co amsterdam
cd ../messageservice
git co amsterdam
cd ..
```

4. In order to remove the need to install A&AI, comment out as follows in msgrtr src/main/java/com/att/nsa/cambria/service/impl/TopicServiceImpl.java:

```
diff --git a/src/main/java/com/att/nsa/cambria/service/impl/TopicServiceImpl.java b/src/main/java/com/att
/nsa/cambria/service/impl/TopicServiceImpl.java
index c12be2f..d11dda0 100644
--- a/src/main/java/com/att/nsa/cambria/service/impl/TopicServiceImpl.java
+++ b/src/main/java/com/att/nsa/cambria/service/impl/TopicServiceImpl.java
@@ -196,7 +196,7 @@ public class TopicServiceImpl implements TopicService {
                String appName=dmaapContext.getRequest().getHeader("AppName");
                String enfTopicName= com.att.ajsc.beans.PropertiesMapBean.getProperty(CambriaConstants.
msgRtr_prop,"enforced.topic.name.AAF");

-               if(user != null)
+               /*if(user != null)
                {
                        key = user.getKey();

@@ -214,7 +214,7 @@ public class TopicServiceImpl implements TopicService {
                }

                //else if (user==null && (null==dmaapContext.getRequest().getHeader("Authorization") &&
null == dmaapContext.getRequest().getHeader("cookie")) ) {
-                       else if (user == null &&  null==dmaapContext.getRequest().getHeader
("Authorization")     &&
+               else if (user == null &&  null==dmaapContext.getRequest().getHeader("Authorization")
&&
                                                (null == appName  &&  null == dmaapContext.getRequest().
getHeader("cookie"))) {
                        LOGGER.error("Failed to create topic"+topicBean.getTopicName()+", Authentication
failed.");

@@ -223,7 +223,7 @@ public class TopicServiceImpl implements TopicService {
                                                errorMessages.getCreateTopicFail()+" "+errorMessages.
getNotPermitted1()+" create "+errorMessages.getNotPermitted2());
                        LOGGER.info(errRes.toString());
```

```
                        throw new DMaaPAccessDeniedException(errRes);
-               }
+                       }*/

                if (user == null &&  (null!=dmaapContext.getRequest().getHeader("Authorization") ||
                                        null != dmaapContext.getRequest().getHeader("cookie"))) {
@@ -243,7 +243,7 @@ public class TopicServiceImpl implements TopicService {
                        permission = mrFactoryVal+nameSpace+"|create";
                        DMaaPAAFAuthenticator aaf = new DMaaPAAFAuthenticatorImpl();

-                       if(!aaf.aafAuthentication(dmaapContext.getRequest(), permission))
+                       /*if(!aaf.aafAuthentication(dmaapContext.getRequest(), permission))
                        {

                                LOGGER.error("Failed to create topic"+topicBean.getTopicName()+",
Authentication failed.");
@@ -254,7 +254,7 @@ public class TopicServiceImpl implements TopicService {
                                LOGGER.info(errRes.toString());
                                throw new DMaaPAccessDeniedException(errRes);

-                       }else{
+                               }else{*/
                                // if user is null and aaf authentication is ok then key should be ""
                                //key = "";
                                /**
@@ -264,7 +264,7 @@ public class TopicServiceImpl implements TopicService {
                                key = dmaapContext.getRequest().getUserPrincipal().getName().toString();
                                LOGGER.info("key ==================== "+key);

-                       }
+                               /*}*/
                }

                try {
@@ -311,7 +311,7 @@ public class TopicServiceImpl implements TopicService {
                LOGGER.info("Deleting topic " + topicName);
                final NsaApiKey user = DMaaPAuthenticatorImpl.getAuthenticatedUser(dmaapContext);

-               if (user == null && null!=dmaapContext.getRequest().getHeader("Authorization")) {
+               /*if (user == null && null!=dmaapContext.getRequest().getHeader("Authorization")) {
                        LOGGER.info("Authenticating the user, as ACL authentication is not provided");
 //                     String permission = "com.att.dmaap.mr.topic"+"|"+topicName+"|"+"manage";
                        String permission = "";
@@ -331,7 +331,7 @@ public class TopicServiceImpl implements TopicService {
                        }


-               }
+                       }*/

                final Broker metabroker = getMetaBroker(dmaapContext);
                final Topic topic = metabroker.getTopic(topicName);
```

5. Build the msgrtr, skipping tests if you wish. Once the build completes, we're finished in msgrtr.

```
cd msgrtr
mvn clean install
```

6. In messageservice, the start-kafka.sh script breaks with the latest version of Scala and Kafka unless a new line is added to the end of the server.properties file. Edit the messageservice src/main/resources/docker-compose/start-kafka.sh as follows:

```
cd ../messageservice
diff --git a/src/main/resources/docker-compose/start-kafka.sh b/src/main/resources/docker-compose/start-
kafka.sh
index 87047ad..e1bf99c 100644
--- a/src/main/resources/docker-compose/start-kafka.sh
+++ b/src/main/resources/docker-compose/start-kafka.sh
@@ -47,6 +47,9 @@ if [[ -z "$KAFKA_ADVERTISED_HOST_NAME" && -n "$HOSTNAME_COMMAND" ]]; then
     export KAFKA_ADVERTISED_HOST_NAME=$(eval $HOSTNAME_COMMAND)
 fi

+# The replacement below fails if there is not a new line at the end of the $KAFKA_HOME/config/server.
properties file
+echo "" >> $KAFKA_HOME/config/server.properties
+
 for VAR in `env`
 do
   if [[ $VAR =~ ^KAFKA_ && ! $VAR =~ ^KAFKA_HOME ]]; then
```

7. Ensure that the version of msgrtr dependency used in messageservice pom.xml is the local one created above, check your pom.xml in messageservice, specifically you must specify if msgrtr is a SNAPSHOT version. for example, see below:

```
        <dependency>
            <groupId>org.onap.dmaap.messagerouter.msgrtr</groupId>
            <artifactId>msgrtr</artifactId>
            <version>1.0.1-SNAPSHOT</version>
```

8. Build the message service, skipping tests if you wish.

```
mvn clean install
```

9. Ensure Docker is running on your machine. Clear down all relevant containers and images in your docker that might interfere with DMaaP. Use the commands below WITH CARE!

```
docker ps -a | cut -f1 -d' ' | awk '{printf("docker rm %s\n", $1)}'
docker images | cut -c45-60 | grep -v ubuntu | awk '{printf("docker rmi --force %s\n", $1)}'
```

10. In messageservice, copy the "appl" directory to target/classes/docker. This copies the configuration for DMaaP into the location in which we will build the docker images and the docker-compose setup.

```
pwd local-dmaap/messageservice
cp -pr target/swm/package/nix/dist_files/appl target/classes/docker
```

11. In messageservice, copy the "startup.sh" script target/classes/docker. This is the startup shell script for the DMaaP message service in Docker.

```
cp target/swm/package/nix/dist_files/startup.sh target/classes/docker
```

12. If /var/tmp/MsgRtr.properties file or directory exists on /var/tmp,delete it. This is the file where we will specify the local properties for Kafka /Zookeeper for DMaaP.

```
rm -fr /var/tmp/MsgRtrApi.properties
```

13. Copy the template MsgRtr properties to /var/tmp

```
cp bundleconfig-local/etc/appprops/MsgRtrApi.properties /var/tmp
```

14. Edit /var/tmp/MsgRtrApi.properties as follows to put in the host names and ports for Kafka and Zookeeper in our docker-compose setup:

```
0c40
< config.zk.servers=<zookeeper_host>
---
> config.zk.servers=zookeeper
52c52
< kafka.metadata.broker.list=<kafka_host>:<kafka_port>
---
> kafka.metadata.broker.list=kafka:9092
```

15. Build the adapted DMaaP docker image, it will be tagged as dmaap:localadapt

```
cd target/classes/docker
docker build -t dmaap:localadapt .
```

16. Now we prepare the docker-compose setup for DMaaP. Go to messageservice/target/classes/docker-compose and edit Dockerfile, changing the KAFKA_VERSION to 1.0.0 and the SCALA_VERSION to 2.12

```
cd ../docker-compose
diff Dockerfile.orig Dockerfile
7c7
< ENV KAFKA_VERSION="0.8.1.1" SCALA_VERSION="2.9.2"
---
> ENV KAFKA_VERSION="1.0.0" SCALA_VERSION="2.12"
```

17. Edit docker-compose.yml to change the name of th DMaaP docker image to use our locally adapted DMaaP image:

```
diff docker-compose.yml.orig docker-compose.yml
21c21
<     image: attos/dmaap
---
>     image: dmaap:localadapt
```

18. On macOS only, edit docker-compose.yml to change var/tmp to /private/var/tmp (ensure you have /private/var/tmp shared with Docker on macOS).
19. Start DMaaP, crossing your fingers really tightly!

```
docker-compose up
```

20. Check that DMaaP is running with a curl request to list topics

```
curl  --request GET localhost:3904/topics

ANSWER
{"topics": []}
```

21. Create a topic called MyTopic

```
 curl \
>     --header "Content-type: application/json" \
>     --request POST \
>     --data '{
>   "topicName":"MyTopic",
>   "partitionCount":"1",
>   "replicationCount":"1",
>   "transactionEnabled":"false"
> }' \
> http://localhost:3904/topics/create

ANSWER:
{
    "owner": "",
    "readerAcl": {
        "enabled": true,
        "users": []
    },
    "name": "MyTopic",
    "description": "",
    "writerAcl": {
        "enabled": true,
        "users": []
    }
}
```

22. List the topics again, our topic should now be there:

```
curl  --request GET localhost:3904/topics

ANSWER:
{"topics": [
    "msgrtr.apinode.metrics.dmaap",
    "MyTopic"
]}
```

Related articles

- model-driven DMaaP Agent
- DMaaP Edge Deployment
- DCAEGEN2 architecture of policy-handling by DCAE-controller
- Cloud Agnostic Intent and Mappings
- ONAP Beijing: Understanding the vFWCL use-case mechanism