Anomaly Detection Use Case

Introduction:

Anomaly Detection (AD), QoE Predictor (QP) and Traffic Steering xApp's are three components in Anomaly Detection use case in Near RT RIC platform. AD xApp aims to detect anomalous UE's present in network. It communicates with Traffic Steering xApp via RMR and send Anomalous UE's information to TS. TS request QP xApp to predict throughput on given anomalous UE list. QP xApp use Machine learning algorithm to predict throughput for a UE in given cell and send that to TS. TS xApp collaborate with AD and QP to gain knowledge on current and neighboring cells and performs a handover to a cell having maximum throughput(if needed).

Features:

AD:

- AD xApp is a component in near RIC platform that communicates with Traffic Steering xApp
- It create a connection to InfluxDB and fetch continuously latest stream of UE data. Also perform write data method for predicted outcomes into InfluxDB
- Use machine learning algorithm to detect anomalous users and find out the degradation type
- Send anomalous UE's along with degradation type to Traffic Steering xApp via RMR

QP:

- It creates a connection to InfluxDB to fetch and write UE and Cell metrics
- QP process a UE ID and fetch latest relevant UE and cell metrics from InfluxDB as and when it gets a prediction request from TS
- Use machine learning algorithm to predict throughput in neighboring cells for a given anomalous users
- Send these predictions to TS xApp

TS:

- TS xApp expect anomalous users list from AD as and when AD detects
- Send prediction request to QP for these anomalous users
- Find cell to handover(if needed) and perform the handover

Architecture:

Prerequisites:

Near-RT RIC platform setup is required. Link for Near-RT RIC installation.

InfluxDB component needs to be deployed (part of Near-RT RIC installation)

Traffic Steering (TS) xApp needs to be build, deployed and onboarded on the Near RT RIC platform

AD xApp needs to be build, deployed and onboarded on the Near RT RIC platform

QP xApp needs to be build, deployed and onboarded on the Near RT RIC platform

InfluxDB :

Onetime setup for InfluxDB

Once Kubernetes setup is done, we have to create PersistentVolume through the storage class for the InfluxDB database. Mentioned below is one time process that should be followed before deploying the InfluxDB in ricplt namespace.

```
# User has to check the following namespace exist or not using
% kubectl get ns ricinfra

# If the namespace doesn't exist, then create it using:
% kubectl create ns ricinfra

% helm install stable/nfs-server-provisioner --namespace ricinfra --name nfs-release-1
% kubectl patch storageclass nfs -p '{"metadata": {"annotations":{"storageclass.kubernetes.io/is-default-class":"true"}}}'
% sudo apt install nfs-common
```

Onboarding and Deployment of AD, QP and TS xApp:

a) xApp's can be cloned from Gerrit using the following command:

AD : git clone "https://gerrit.o-ran-sc.org/r/ric-app/ad"

TS : git clone "https://gerrit.o-ran-sc.org/r/ric-app/ts"

QP : git clone "https://gerrit.o-ran-sc.org/r/ric-app/qp"

build the images locally after cloning it.

OR
b) build and push the docker image from nexus repository :

AD:

```
docker build -t nexus3.o-ran-sc.org:10002/o-ran-sc/ric-app-ad:0.0.2 .
docker push  nexus3.o-ran-sc.org:10002/o-ran-sc/ric-app-ad:0.0.2
```

TS:

```
docker build -t nexus3.o-ran-sc.org:10002/o-ran-sc/ric-app-ts:1.0.1 .
docker push  nexus3.o-ran-sc.org:10002/o-ran-sc/ric-app-ts:1.0.1
```

QP:

```
docker build -t nexus3.o-ran-sc.org:10002/o-ran-sc/ric-app-qp:0.0.3 .
docker push  nexus3.o-ran-sc.org:10002/o-ran-sc/ric-app-qp:0.0.3
```

Onboarding xApp's:

Step:1

Each of these xApp's have a descriptor in their Gerrit repo under the xapp-descriptor/ directory. None of them have xapp specific controls and therefore no individual json schema.
Here are the URLs which can be included in HTTP POST call to onboard tool.

AD : https://gerrit.o-ran-sc.org/r/gitweb?p=ric-app/ad.git;a=blob_plain;f=xapp-descriptor/config.json;hb=HEAD
TS : https://gerrit.o-ran-sc.org/r/gitweb?p=ric-app/ts.git;a=blob_plain;f=xapp-descriptor/config.json;hb=HEAD

QP : https://gerrit.o-ran-sc.org/r/gitweb?p=ric-app/qp.git;a=blob_plain;f=xapp-descriptor/config.json;hb=HEAD

Step:2
Here we are preparing for API calls into the xApp On-Boarder by providing the locations of the xApp descriptors.

AD : echo '{"config-file.json_url": "https://gerrit.o-ran-sc.org/r/gitweb?p=ric-app/ad.git;a=blob_plain;f=xapp-descriptor/config.json;hb=HEAD" }' > onboard.ad.url
TS : echo '{"config-file.json_url": "https://gerrit.o-ran-sc.org/r/gitweb?p=ric-app/ts.git;a=blob_plain;f=xapp-descriptor/config.json;hb=HEAD" }' > onboard.ts.url

QP : echo '{"config-file.json_url": "https://gerrit.o-ran-sc.org/r/gitweb?p=ric-app/qp.git;a=blob_plain;f=xapp-descriptor/config.json;hb=HEAD" }' > onboard.qp.url

Step:3
Invoke the API calls into the xApp On-boarder, providing it the locations of the xApp descriptors.

AD : curl --location --request POST "http://$(hostname):32080/onboard/api/v1/onboard/download" --header 'Content-Type: application/json' --data-binary "@./onboard.ad.url"
TS : curl --location --request POST "http://$(hostname):32080/onboard/api/v1/onboard/download" --header 'Content-Type: application/json' --data-binary "@./onboard.ts.url"

QP : curl --location --request POST "http://$(hostname):32080/onboard/api/v1/onboard/download" --header 'Content-Type: application/json' --data-binary "@./onboard.qp.url"

Checking the on-boarded charts:
curl --location --request GET "http://$(hostname):32080/onboard/api/v1/charts"

Deploying xApp:
Deploy the xApp's by invoking the xApp Manager's API. Once receiving the deploy API call, the xApp Manager will make API call into Helm/Kubernetes to deploy the xApp's Helm chart.
The Routing Manager is also involved if the xApp needs to process RMR messages, – it will complete the routes and send out route updates.

AD : curl --location --request POST "http://$(hostname):32080/appmgr/ric/v1/xapps" --header 'Content-Type: application/json' --data-raw '{"xappName": "ad"}'

TS : curl --location --request POST "http://$(hostname):32080/appmgr/ric/v1/xapps" --header 'Content-Type: application/json' --data-raw '{"xappName": "trafficxapp"}'

QP : curl --location --request POST "http://$(hostname):32080/appmgr/ric/v1/xapps" --header 'Content-Type: application/json' --data-raw '{"xappName": "qp"}'

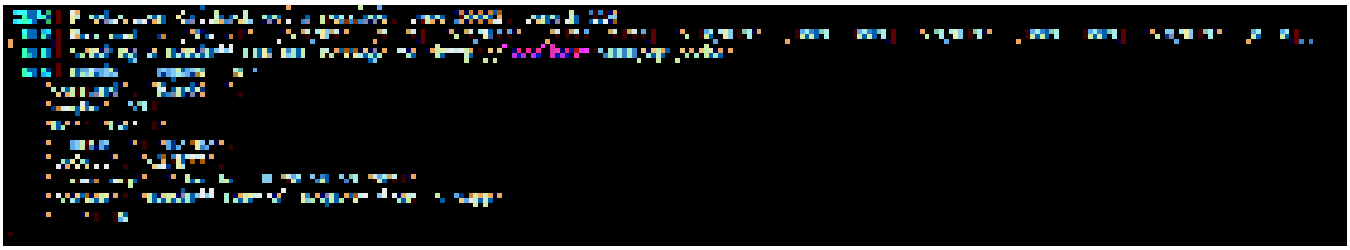
Verification:

AD : Check for logs in pods where AD is running, It should have following logs(It might take some time for AD to detect an anomalous user since ideally majority of users will be normal in network and to get motioned logs below):

```
[INFO] Sending Anomalous UE to TS
[INFO] Message to TS: message sent Successfully
[INFO] Received acknowldgement from TS (TS_ANOMALY_ACK): {'payload': b'[{"du-id": 1002, "ue-id": "Car-1", "measTimeStampRf": 1625775890611, "De
gradation": "Throughput RSRP RSRQ"}]', 'payload length': 108, 'message type': 30004, 'subscription id': -1, 'transaction id': b'150fd5f4e02e11e
b80ec2a532cd4256f', 'message state': 0, 'message status': 'RMR_OK', 'payload max size': 3136, 'meid': b'', 'message source': 'service-ricxapp-t
rafficxapp-rmr.ricxapp:4560', 'errno': 0}
```

TS : Check for logs in pods where TS is running, It should have following logs:

```
[INFO] AD Callback got a message, type=30003, length=115
[INFO] Payload is [{"du-id": 1002, "ue-id": "Car-1", "measTimeStampRf": 1625775966081, "Degradation": "Throughput RSRP RSRQ RSSINR"}]
[INFO] Prediction Request length=30, payload={"UEPredictionSet": ["Car-1"]}
```



QP : Check for logs in pods where QP is running, It should have following logs:

```
{"ts": 1625778810175, "crit": "DEBUG", "id": "ricxappframe.xapp_frame", "mdc": {}, "msg": "run: invoking msg handler on type 30000"}
{"ts": 1625778810176, "crit": "DEBUG", "id": "qp.main", "mdc": {}, "msg": "predict handler received payload b'{\"UEPredictionSet\": [\"Car-1\"]
}'"}
{"ts": 1625778810767, "crit": "DEBUG", "id": "qp.main", "mdc": {}, "msg": "Sending message to ts : {\"Car-1\": {\"c2/B2\": [2, 2], \"c2/N77\":
[2127, 2127], \"c10/B13\": [1063, 1063], \"c3/B13\": [1063, 1063], \"c9/B13\": [0, 0]}}"}
{"ts": 1625778810767, "crit": "DEBUG", "id": "qp.main", "mdc": {}, "msg": "predict handler: sent message successfully"}
```

Deleting xApp:

AD : curl --location --request DELETE "http://$(hostname):32080/appmgr/ric/v1/xapps/ad" --header 'Content-Type: application/json'

TS : curl --location --request DELETE "$(hostname):32080/appmgr/ric/v1/xapps/trafficxapp" --header 'Content-Type: application/json'

QP : curl --location --request DELETE "$(hostname):32080/appmgr/ric/v1/xapps/qp" --header 'Content-Type: application/json'

Notes:

1. All three xApp's needs to be deployed in following sequence AD TS QP for end to end testing of AD use case
2. Names of the xApp to be deployed must match with what the on-boarder has
3. AD uses message types TS_ANOMALY_UPDATE to send anomalous UE's and TS_ANOMALY_ACK for acknowledgment from TS on receiving same message successfully
4. QP send UEPredicitonSet message to TS and expects predicted throughput message callback