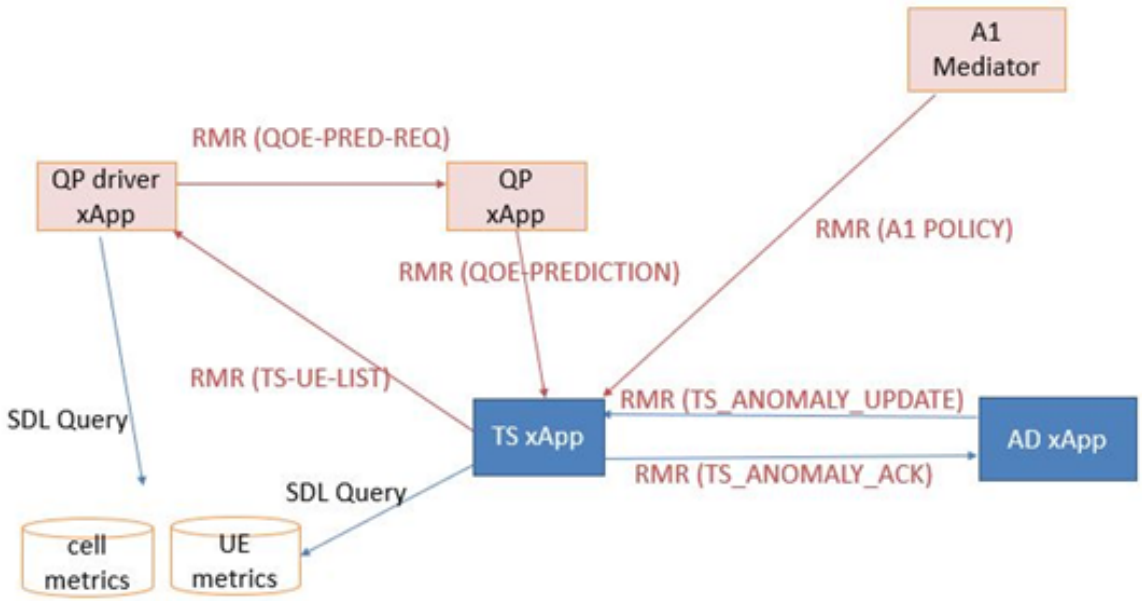
AD xApp Flows

Anomaly Detection (AD) xApp is used to detect the anomalous UEs present in the network.
It works with Traffic Steering xApp.

Here is the RMR communication diagram among the xApps:



Demo Video

Your browser does not support the HTML5 video element

Delivery Status

DO NOT EDIT

Only the Integration PTL should edit the delivery status table to track the release validation progress

		OTF	OAM	NONRTRIC	RICP	RICAPP	O-DU	O-CU	Test Result	Notes
Deployment Artifacts	Docker Container									AD container is locally built.
	xApp Descriptors									

E2E Flows	TS_ANOMALY and TS_ANOMALY_ACK									

```
app:4560', 'errno': 0}
val:  b'[{"UEID": 12359, "MeasTimestampRF": "2020-12-07 14:52:31.025943"}]'
TS_ANOMALY_ACK: {'payload': b'[{"UEID": 12367, "MeasTimestampRF": "2020-12-07 14:52:25.776482"}]', 'payload len
': b'd8c4c506389b11eb832b060a7e949426', 'message state': 0, 'message status': 'RMR_OK', 'payload max size': 313
app:4560', 'errno': 0}
val:  b'[{"UEID": 12402, "MeasTimestampRF": "2020-12-07 14:52:10.816951"}]'
TS_ANOMALY_ACK: {'payload': b'[{"UEID": 12359, "MeasTimestampRF": "2020-12-07 14:52:31.025943"}]', 'payload len
': b'd9696ca8389b11ebbf2e060a7e949426', 'message state': 0, 'message status': 'RMR_OK', 'payload max size': 313
app:4560', 'errno': 0}
1607352761 1/RMR [INFO] sends: ts=1607352761 src=service-ricxapp-ad-rmr.ricxapp:4560 target=service-ricplt-a1me
1607352761 1/RMR [INFO] sends: ts=1607352761 src=service-ricxapp-ad-rmr.ricxapp:4560 target=service-ricplt-subm
1607352761 1/RMR [INFO] sends: ts=1607352761 src=service-ricxapp-ad-rmr.ricxapp:4560 target=service-ricxapp-tra
1607352761 1/RMR [INFO] sends: ts=1607352761 src=service-ricxapp-ad-rmr.ricxapp:4560 target=service-ricplt-e2mg
1607352761 1/RMR [INFO] sends: ts=1607352761 src=service-ricxapp-ad-rmr.ricxapp:4560 target=service-ricxapp-tra
val:  b'[{"UEID": 12439, "MeasTimestampRF": "2020-12-07 14:51:52.644308"}]'
TS_ANOMALY_ACK: {'payload': b'[{"UEID": 12402, "MeasTimestampRF": "2020-12-07 14:52:10.816951"}]', 'payload len
': b'dab31564389b11eba140060a7e949426', 'message state': 0, 'message status': 'RMR_OK', 'payload max size': 313
app:4560', 'errno': 0}
val:  b'[{"UEID": 12355, "MeasTimestampRF": "2020-12-07 14:52:38.499258"}]'
TS_ANOMALY_ACK: {'payload': b'[{"UEID": 12439, "MeasTimestampRF": "2020-12-07 14:51:52.644308"}]', 'payload len
': b'db579308389b11eb97ff060a7e949426', 'message state': 0, 'message status': 'RMR_OK', 'payload max size': 313
app:4560', 'errno': 0}
val:  b'[{"UEID": 12368, "MeasTimestampRF": "2020-12-07 14:52:32.821644"}]'
TS_ANOMALY_ACK: {'payload': b'[{"UEID": 12355, "MeasTimestampRF": "2020-12-07 14:52:38.499258"}]', 'payload len
': b'dca04940389b11eb8a69060a7e949426', 'message state': 0, 'message status': 'RMR_OK', 'payload max size': 313
app:4560', 'errno': 0}
val:  b'[{"UEID": 12366, "MeasTimestampRF": "2020-12-07 14:52:34.940748"}]'
TS_ANOMALY_ACK: {'payload': b'[{"UEID": 12368, "MeasTimestampRF": "2020-12-07 14:52:32.821644"}]', 'payload len
': b'd46b978389b11ebba7d060a7e949426', 'message state': 0, 'message status': 'RMR_OK', 'payload max size': 313
app:4560', 'errno': 0}
val:  b'[{"UEID": 12350, "MeasTimestampRF": "2020-12-07 14:52:44.339455"}]'
TS_ANOMALY_ACK: {'payload': b'[{"UEID": 12366, "MeasTimestampRF": "2020-12-07 14:52:34.940748"}]', 'payload len
': b'ddea3bba389b11ebb756060a7e949426', 'message state': 0, 'message status': 'RMR_OK', 'payload max size': 313
app:4560', 'errno': 0}
val:  b'[{"UEID": 12377, "MeasTimestampRF": "2020-12-07 14:52:31.364605"}]'
TS_ANOMALY_ACK: {'payload': b'[{"UEID": 12350, "MeasTimestampRF": "2020-12-07 14:52:44.339455"}]', 'payload len
': b'de8f9da4389b11eb884c060a7e949426', 'message state': 0, 'message status': 'RMR_OK', 'payload max size': 313
app:4560', 'errno': 0}
val:  b'[{"UEID": 12372, "MeasTimestampRF": "2020-12-07 14:52:35.043930"}]'
TS_ANOMALY_ACK: {'payload': b'[{"UEID": 12377, "MeasTimestampRF": "2020-12-07 14:52:31.364605"}]', 'payload len
': b'df347d06389b11eb94bf060a7e949426', 'message state': 0, 'message status': 'RMR_OK', 'payload max size': 313
app:4560', 'errno': 0}
val:  b'[{"UEID": 12350, "MeasTimestampRF": "2020-12-07 14:52:47.584468"}]'
TS_ANOMALY_ACK: {'payload': b'[{"UEID": 12372, "MeasTimestampRF": "2020-12-07 14:52:35.043930"}]', 'payload len
': b'dfd94afe389b11ebadd3060a7e949426', 'message state': 0, 'message status': 'RMR_OK', 'payload max size': 313
app:4560', 'errno': 0}
val:  b'[{"UEID": 12354, "MeasTimestampRF": "2020-12-07 14:52:46.572806"}]'
TS_ANOMALY_ACK: {'payload': b'[{"UEID": 12350, "MeasTimestampRF": "2020-12-07 14:52:47.584468"}]', 'payload len
': b'e074f5c389b11eb9d2806a7e949426', 'message state': 0, 'message status': 'RMR_OK', 'payload max size': 313
app:4560', 'errno': 0}
```

Onboarding and Deployment of xApps:

The Use case involves two different xApps:
a) Traffic Steering (TS) xApp (AT&T)
b) Anomaly Detection (AD) xApp (HCL)

xApps can be cloned from nexus repository using the following command:
TS xApp:
    git clone "https://gerrit.o-ran-sc.org/r/ric-app/ts"
AD xApp:
    git clone "https://gerrit.o-ran-sc.org/r/ric-app/ad"

Currently, we are building the images locally after cloning it.
Once the Staging and Release images are available, we can use the below commands.
Build and push the docker image for both AD xApp and TS xApp using the following:

TS xApp:
  docker build -t nexus3.o-ran-sc.org:10002/o-ran-sc/ric-app-ts:1.0.13 .
  docker push nexus3.o-ran-sc.org:10002/o-ran-sc/ric-app-ts:1.0.13

AD xApp:
  docker build -t nexus3.o-ran-sc.org:10002/o-ran-sc/ric-app-ad:0.0.1 .
  docker push nexus3.o-ran-sc.org:10002/o-ran-sc/ric-app-ad:0.0.1

Each of these xApps 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 for each which can be included in HTTP POST call to onboard tool.
TS xApp:
https://gerrit.o-ran-sc.org/r/gitweb?p=ric-app/ts.git;a=blob_plain;f=xapp-descriptor/config.json;hb=HEAD
AD xApp:
https://gerrit.o-ran-sc.org/r/gitweb?p=ric-app/ad.git;a=blob_plain;f=xapp-descriptor/config.json;hb=HEAD

It is required to deploy TS xApp followed by AD xApp.

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

Onboarding xApps:
Invoke the API calls into the xApp On-boarder, providing it the locations of the xApp descripters.
curl --location --request POST "http://$(hostname):32080/onboard/api/v1/onboard/download" --header 'Content-Type: application/json' --data-binary "@./onboard.ts.url"
curl --location --request POST "http://$(hostname):32080/onboard/api/v1/onboard/download" --header 'Content-Type: application/json' --data-binary "@./onboard.ad.url"

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

Deploying xApp:
Deploy the xApps by invoking the xApp Manager's API.
Note that the names of the xApp to be deployed must match with what the on-boarder has.
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.

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

Note: Use the latest Routing Manager supporting the new message types (TS_ANOMALY_UPDATE and TS_ANOMALY_ACK).