Release F - Run in Docker

This page describes how to get the release F version of Non-RT RIC up and running locally with three separate A1 simulator (previously called Near-RT RIC A1 Interface) docker containers providing STD_1.1.3, STD_2.0.0 and OSC_2.1.0 versions of the A1 interface.

All components of the Non-RT RIC run as docker containers and communicate via a private docker network with container ports, ports also available on localhost. Details of the architecture can be found from Release F page.

Project Requirements

- Docker and docker-compose (latest)
- kubectl with admin access to kubernetes (minikube, docker-desktop kubernetes etc) - this is only applicable when running the Helm Manager
- helm with access to kubernetes - this is only applicable when running the Helm Manager example operations

Images

The images used for running the Non-RT RIC can be selected from the table below depending on if the images are built manually (snapshot image) or if release images shall be used.

In general, there is no need to build the images manually unless there are code changes made by the user, so release images should be used. Instruction on how to build all components, see. Release F - Build.

The run commands throughout this page uses the **release** images and tags. Replace the release images/tags in the container run commands in the instructions if manually-built snapshot images are desired.

F Maintenance Release images

Component	Image	Tag
A1 Policy Management Service	nexus3.o-ran-sc.org:10002/o-ran-sc/nonrtric-plt-a1policymanagementservice	2.4.1
Information Coordinator Service	nexus3.o-ran-sc.org:10002/o-ran-sc/nonrtric-plt-informationcoordinatorservice	1.3.1
NONRTRIC Control Panel	nexus3.o-ran-sc.org:10002/o-ran-sc/nonrtric-controlpanel	2.3.0
Gateway	nexus3.o-ran-sc.org:10002/o-ran-sc/nonrtric-gateway	1.0.0
A1-Simulator	nexus3.o-ran-sc.org:10002/o-ran-sc/a1-simulator	2.3.1
R-APP Catalogue	nexus3.o-ran-sc.org:10002/o-ran-sc/nonrtric-plt-rappcatalogue	1.1.0
DMaaP Adapter	nexus3.o-ran-sc.org:10002/o-ran-sc/nonrtric-plt-dmaapadapter	1.1.1
DMaaP Mediator	nexus3.o-ran-sc.org:10002/o-ran-sc/nonrtric-plt-dmaapmediatorproducer	1.1.0
Helm Manager	nexus3.o-ran-sc.org:10002/o-ran-sc/nonrtric-plt-helmmanager	1.2.0
Auth Token Fetcher	nexus3.o-ran-sc.org:10002/o-ran-sc/nonrtric-plt-auth-token-fetch	1.0.0
O-DU Slice Assurance	nexus3.o-ran-sc.org:10002/o-ran-sc/nonrtric-rapp-ransliceassurance	1.1.1
O-DU Slice Assurance	nexus3.o-ran-sc.org:10002/o-ran-sc/nonrtric-rapp-ransliceassurance-icsversion	1.0.0

O-RU FH Recovery	nexus3.o-ran-sc.org:10002/o-ran-sc/nonrtric-rapp-orufhrecovery	1.1.0
O-RU FH Recovery	nexus3.o-ran-sc.org:10002/o-ran-sc/nonrtric-rapp-orufhrecovery-consumer	1.1.0

Note: A version of this table appears Integration&Testing - F Release Docker Image List - NONRTRIC (F-Release). This is the authoritative version!

Ports

The following ports will be allocated and exposed to localhost for each component. If other port(s) are desired, then the ports need to be replaced in the container run commands in the instructions further below.

Component	Port exposed to localhost (http/https)
A1 Policy Management Service	8081/8443
A1 Simulator (previously called Near-RT RIC A1 Interface)	8085/8185, 8086/8186, 8087/8187
Information Coordinator Service	8083/8434
Non-RT RIC Control Panel	8080/8880
SDNC A1-Controller	8282/8443
Gateway	9090 (only http)
App Catalogue Service	8680/8633
Helm Manager	8112 (only http)
Dmaap Mediator Producer	9085/9185
Dmaap Adapter Service	9087/9187

Prerequisites

The containers need to be connected to docker network in order to communicate with each other.

Create a private docker network. If another network name is used, all references to 'nonrtric-docker-net' in the container run commands below need to be replaced.

```
docker network create nonrtric-docker-net
```

Run the A1 Policy Management Service Docker Container

To support local test with three separate A1 simulator (previously called Near-RT RIC A1 Interface) instances, each running a one of the three available A1 Policy interface versions:

- Create an application_configuration.json file with the configuration below. This will configure the policy management service to use the simulators for the A1 interface
- Note: Any defined ric names must match the given docker container names in near-RT RIC simulator startup, see Run A1 (previously called Near-RT RIC A1 Interface) Simulator Docker Containers
- The application supports both REST and DMAAP interface. REST is always enabled but to enable DMAAP (message exchange via message-router) additional config is needed. The examples below uses REST over http.

The policy management service can be configure with or without a A-Controller. Choose the appropriate configuration below.

This file is for running without the SDNC A1-Controller

application_configuration.json

```
{
   "config": {
      "ric": [
         {
            "name": "ric1",
            "baseUrl": "http://ric1:8085/",
            "managedElementIds": [
               "kista_1",
               "kista_2"
            ]
         },
         {
            "name": "ric2",
            "baseUrl": "http://ric2:8085/",
            "managedElementIds": [
               "kista_3",
               "kista_4"
            ]
         },
         {
            "name": "ric3",
            "baseUrl": "http://ric3:8085/",
            "managedElementIds": [
               "kista_5",
               "kista_6"
            ]
         }
      ]
   }
}
```

This file is for running with the SDNC A1-Controller.

application_configuration.json

```
{
   "config": {
       "controller": [
             {
             "name": "a1controller",
             "baseUrl": "https://a1controller:8443",
             "userName": "admin",
             "password": "Kp8bJ4SXszM0WXlhak3eHlcse2gAw84vaoGGmJvUy2U"
         }
      ],
      "ric": [
         {
            "name": "ric1",
            "baseUrl": "http://ric1:8085/",
            "controller": "a1controller",
            "managedElementIds": [
               "kista_1",
               "kista_2"
            ]
         },
         {
            "name": "ric2",
            "baseUrl": "http://ric2:8085/",
            "controller": "a1controller",
            "managedElementIds": [
               "kista_3",
               "kista_4"
            ]
         },
         {
            "name": "ric3",
            "baseUrl": "http://ric3:8085/",
            "controller": "a1controller",
            "managedElementIds": [
               "kista_5",
               "kista_6"
            ]
         }
      ]
   }
}
```

To enable the also the optional DMaaP interface, add the following config (same level as the "ric" entry) to application_configuration.json.

Be sure to update http/host/port below to match the configuration of the used message router.

A1 Policy Management Service DMaaP interface is deprecated

```
Note: The DMaaP interface for the A1 Policy Management Service is now deprecated
```

optional dmaap config in application_configuration.json

```
  ...
  "streams_publishes": {
    "dmaap_publisher": {
      "type": "message-router",
      "dmaap_info": {
        "topic_url": "http://dmaap-mr:3904/events/A1-POLICY-AGENT-WRITE"
      }
    }
  },
  "streams_subscribes": {
    "dmaap_subscriber": {
      "type": "message-router",
      "dmaap_info": {
        "topic_url": "http://dmaap-mr:3904/events/A1-POLICY-AGENT-READ/users/policy-agent?
timeout=15000&limit=100"
      }
    }
  },
  ...
```

Start the container with the following command. Replace "<absolute-path-to-file>" with the the path to the created configuration file in the command. The configuration file is mounted to the container. There will be WARN messages appearing in the log until the simulators are started.

```
docker run --rm -v <absolute-path-to-file>/application_configuration.json:/opt/app/policy-agent/data
/application_configuration.json -p 8081:8081 -p 8433:8433 --network=nonrtric-docker-net --name=policy-agent-
container nexus3.o-ran-sc.org:10002/o-ran-sc/nonrtric-plt-a1policymanagementservice:2.4.1
```

Wait 1 minute to allow the container to start and to read the configuration. Then run the command below another terminal. The output should match the configuration in the file - all three rics (ric1, ric2 and ric3) should be included in the output. Note that each ric has the state "UNAVAILABLE" until the simulators are started.

Note: If the policy management service is started with config for the SDNC A1 Controller (the second config option), do the steps described in section Run the A1 Controller Docker Container below before proceeding.

```
curl localhost:8081/a1-policy/v2/rics
```

Expected output (not that all simulators - ric1,ric2 and ric3 will indicate "state":"UNAVAILABLE" until the simulators has been started in Run A1 (previously called Near-RT RIC A1 Interface) Simulator Docker Containers):

```
{"rics":[{"ric_id":"ric1","managed_element_ids":["kista_1","kista_2"],"policytype_ids":[],"state":"
UNAVAILABLE"},{"ric_id":"ric3","managed_element_ids":["kista_5","kista_6"],"policytype_ids":[],"state":"
UNAVAILABLE"},{"ric_id":"ric2","managed_element_ids":["kista_3","kista_4"],"policytype_ids":[],"state":"
UNAVAILABLE"}]}
```

For troubleshooting/verification purposes you can view/access the full swagger API from url: http://localhost:8081/swagger-ui/index.html?configUrl=/v3/api-docs/swagger-config

Run the SDNC A1 Controller Docker Container (ONAP SDNC)

This step is only applicable if the configuration for the Policy Management Service include the SDNC A1 Controller (second config option), see Run the Policy Management Service Docker Container.

Create the docker compose file - be sure to update image for the a1controller to the one listed for SDNC A1 Controller in the table on the top of this page.

docker-compose.yaml

```
version: '3'

networks:
  default:
    external: true
    name: nonrtric-docker-net

services:
  db:
    image: nexus3.o-ran-sc.org:10001/mariadb:10.5
    container_name: sdncdb
    networks:
      - default
    ports:
      - "3306"
    environment:
      - MYSQL_ROOT_PASSWORD=itsASecret
      - MYSQL_ROOT_HOST=%
      - MYSQL_USER=sdnctl
      - MYSQL_PASSWORD=gamma
      - MYSQL_DATABASE=sdnctl
    logging:
      driver:   "json-file"
      options:
        max-size: "30m"
        max-file: "5"

  a1controller:
    image: nexus3.onap.org:10002/onap/sdnc-image:2.3.2
    depends_on :
      - db
    container_name: a1controller
    networks:
      - default
    entrypoint: ["/opt/onap/sdnc/bin/startODL.sh"]
    ports:
      - 8282:8181
      - 8443:8443
    links:
      - db:dbhost
      - db:sdnctldb01
      - db:sdnctldb02
    environment:
      - MYSQL_ROOT_PASSWORD=itsASecret
      - MYSQL_USER=sdnctl
      - MYSQL_PASSWORD=gamma
      - MYSQL_DATABASE=sdnctl
      - SDNC_CONFIG_DIR=/opt/onap/sdnc/data/properties
      - SDNC_BIN=/opt/onap/sdnc/bin
      - ODL_CERT_DIR=/tmp
      - ODL_ADMIN_USERNAME=admin
      - ODL_ADMIN_PASSWORD=Kp8bJ4SXszM0WXlhak3eHlcse2gAw84vaoGGmJvUy2U
      - ODL_USER=admin
      - ODL_PASSWORD=Kp8bJ4SXszM0WXlhak3eHlcse2gAw84vaoGGmJvUy2U
      - SDNC_DB_INIT=true
      - A1_TRUSTSTORE_PASSWORD=a1adapter
      - AAI_TRUSTSTORE_PASSWORD=changeit
    logging:
      driver:   "json-file"
      options:
        max-size: "30m"
        max-file: "5"
```

Start the SNDC A1 controller with the following command, using the created docker-compose file.

```
docker-compose up
```

Open this url in a web browser to verify that the SDNC A1 Controller is up and running. It may take a few minutes until the url is available.

http://localhost:8282/apidoc/explorer/index.html#/controller%20A1-ADAPTER-API
Username/password: admin/Kp8bJ4SXszM0WXlhak3eHlcse2gAw84vaoGGmJvUy2U

The Karaf logs of A1 controller can be followed e.g. by using command

```
docker exec a1controller sh -c "tail -f /opt/opendaylight/data/log/karaf.log"
```

Run the A1 Simulator (previously called Near-RT RIC A1 Interface) Docker Containers

Start a simulator for each ric defined in in the **application_configuration.json** created in Run the Policy Management Service Docker Container. Each simulator will use one of the currently available A1 interface versions.

Ric1

```
docker run --rm -p 8085:8085 -p 8185:8185 -e A1_VERSION=OSC_2.1.0 -e ALLOW_HTTP=true --network=nonrtric-docker-
net --name=ric1 nexus3.o-ran-sc.org:10002/o-ran-sc/a1-simulator:2.3.1
```

Ric2

```
docker run --rm -p 8086:8085 -p 8186:8185 -e A1_VERSION=STD_1.1.3 -e ALLOW_HTTP=true --network=nonrtric-docker-
net --name=ric2 nexus3.o-ran-sc.org:10002/o-ran-sc/a1-simulator:2.3.1
```

Ric3

```
docker run --rm -p 8087:8085 -p 8187:8185 -e A1_VERSION=STD_2.0.0 -e ALLOW_HTTP=true --network=nonrtric-docker-
net --name=ric3 nexus3.o-ran-sc.org:10002/o-ran-sc/a1-simulator:2.3.1
```

Wait at least one minute to let the policy management service synchronise the rics. Then run the command below another terminal. The output should match the configuration in the file. Note that each ric now has the state "AVAILABLE".

```
curl localhost:8081/a1-policy/v2/rics
```

Expected output - all state should indicated AVAILABLE:

```
{"rics":[{"ric_id":"ric1","managed_element_ids":["kista_1","kista_2"],"policytype_ids":[],"state":"AVAILABLE"},
{"ric_id":"ric3","managed_element_ids":["kista_5","kista_6"],"policytype_ids":[],"state":"AVAILABLE"},
{"ric_id":"ric2","managed_element_ids":["kista_3","kista_4"],"policytype_ids":[""],"state":"AVAILABLE"}]}
```

The simulators using version STD_2.0.0 and OSC_2.1.0 supports policy types. Run the commands below to add one policy types in ric1 and ric3.

Create the file with policy type for ric1

osc_pt1.json

```
{
  "name": "pt1",
  "description": "pt1 policy type",
  "policy_type_id": 1,
  "create_schema": {
    "$schema": "http://json-schema.org/draft-07/schema#",
    "title": "OSC_PT1_0.1.0",
    "description": "QoS policy type",
    "type": "object",
    "properties": {
      "scope": {
        "type": "object",
        "properties": {
          "ueId": {
            "type": "string"
          },
          "qosId": {
            "type": "string"
          }
        },
        "additionalProperties": false,
        "required": [
          "ueId",
          "qosId"
        ]
      },
      "statement": {
        "type": "object",
        "properties": {
          "priorityLevel": {
            "type": "number"
          }
        },
        "additionalProperties": false,
        "required": [
          "priorityLevel"
        ]
      }
    }
  }
}
```

Put the policy type to ric1 - should http response code 201

```
curl -X PUT -v "http://localhost:8085/a1-p/policytypes/123" -H "accept: application/json" \
 -H "Content-Type: application/json" --data-binary @osc_pt1.json
```

Create the file with policy type for ric3

std_pt1.json

```
{
    "policySchema": {
      "$schema": "http://json-schema.org/draft-07/schema#",
      "title": "STD_QOS_0_2_0",
      "description": "STD QOS policy type",
      "type": "object",
      "properties": {
        "scope": {
          "type": "object",
          "properties": {
            "ueId": {
              "type": "string"
            },
            "qosId": {
              "type": "string"
            }
          },
          "additionalProperties": false,
          "required": [
            "ueId",
            "qosId"
          ]
        },
        "qosObjectives": {
          "type": "object",
          "properties": {
            "priorityLevel": {
              "type": "number"
            }
          },
          "additionalProperties": false,
          "required": [
            "priorityLevel"
          ]
        }
      }
    },
    "statusSchema": {
      "$schema": "http://json-schema.org/draft-07/schema#",
      "title": "STD_QOS_0.2.0",
      "description": "STD QOS policy type status",
      "type": "object",
      "properties": {
        "enforceStatus": {
          "type": "string"
        },
        "enforceReason": {
          "type": "string"
        },
        "additionalProperties": false,
        "required": [
          "enforceStatus"
        ]
      }
    }
  }
```

Put the policy type to ric3 - should return http response code 201

```
curl -X PUT -v "http://localhost:8087/policytype?id=std_pt1" -H "accept: application/json"  -H "Content-Type:
application/json" --data-binary @std_pt1.json
```

Wait one minute to let the policy management service synchronise the types with the simulators.

List the synchronised types.

```
curl localhost:8081/a1-policy/v2/policy-types
```

Expected output:

```
{"policytype_ids":["","123","std_pt1"]}
```

Run the Information Coordinator Service Docker Container

Run the following command to start the information coordinator service.

```
docker run --rm -p 8083:8083 -p 8434:8434 --network=nonrtric-docker-net --name=information-service-container
nexus3.o-ran-sc.org:10002/o-ran-sc/nonrtric-plt-informationcoordinatorservice:1.3.1
```

Verify that the Information Coordinator Service is started and responding (response is an empty array).

```
curl localhost:8083/data-producer/v1/info-types
```

Expected output:

```
[ ]
```

For troubleshooting/verification purposes you can view/access the full swagger API from url: http://localhost:8083/swagger-ui/index.html?configUrl=/v3/api-docs/swagger-config

Run the Non-RT RIC Gateway and Control Panel Docker Container

The Gateway exposes the interfaces of the Policy Management Service and the Inform Coordinator Service to a single port of the gateway. This single port is then used by the control panel to access both services.

Create the config file for the gateway.

application.yaml

```
server:
  port: 9090
spring:
  cloud:
    gateway:
      httpclient:
        ssl:
          useInsecureTrustManager: true
        wiretap: true
      httpserver:
        wiretap: true
      routes:
      - id: A1-Policy
        uri: https://policy-agent-container:8433
        predicates:
        - Path=/a1-policy/**
      - id: A1-EI-P
        uri: https://information-service-container:8434
        predicates:
        - Path=/data-producer/**
      - id: A1-EI-C
        uri: https://information-service-container:8434
        predicates:
        - Path=/data-consumer/**
management:
  endpoint:
    gateway:
      enabled: true
  endpoints:
    web:
      exposure:
        include: "gateway,loggers,logfile,health,info,metrics,threaddump,heapdump"
logging:
  level:
    ROOT: ERROR
    org.springframework: ERROR
    org.springframework.cloud.gateway: INFO
    reactor.netty: INFO
  file:
    name: /var/log/nonrtric-gateway/application.log
```

Run the following command to start the gateway. Replace "<absolute-path-to-file>" with the the path to the created application.yaml.

```
docker run --rm -v <absolute-path-to-config-file>/application.yaml:/opt/app/nonrtric-gateway/config/application.
yaml -p 9090:9090 --network=nonrtric-docker-net --name=nonrtric-gateway  nexus3.o-ran-sc.org:10002/o-ran-sc
/nonrtric-gateway:1.0.0
```

Run the following two commands to check that the services can be reached through the gateway

```
curl localhost:9090/a1-policy/v2/rics
```

Expected output

```
{"rics":[{"ric_id":"ric1","managed_element_ids":["kista_1","kista_2"],"policytype_ids":["123"],"state":"
AVAILABLE"},{"ric_id":"ric3","managed_element_ids":["kista_5","kista_6"],"policytype_ids":["std_pt1"],"state":"
AVAILABLE"},{"ric_id":"ric2","managed_element_ids":["kista_3","kista_4"],"policytype_ids":[""],"state":"
AVAILABLE"}]}
```

Second command:

```
curl localhost:9090/data-producer/v1/info-types
```

Expected output:

```
[ ]
```

Create the config file for the control panel.

nginx.conf

```
events{}

http {
    include /etc/nginx/mime.types;
    resolver 127.0.0.11;
    server {
        listen 8080;
        server_name localhost;
        root /usr/share/nginx/html;
        index index.html;
        location /a1-policy/ {
            set $upstream nonrtric-gateway;
            proxy_pass http://$upstream:9090;
        }
        location /data-producer/ {
            set $upstream nonrtric-gateway;
            proxy_pass http://$upstream:9090;
        }
        location /data-consumer/ {
            set $upstream nonrtric-gateway;
            proxy_pass http://$upstream:9090;
        }
        location / {
            try_files $uri $uri/ /index.html;
        }
    }
}
```

Run the following command to start the control panel. Replace "<absolute-path-to-file>" with the the path to the created nginx.conf.

```
docker run --rm -v <absolute-path-to-config-file>/nginx.conf:/etc/nginx/nginx.conf -p 8080:8080 --
network=nonrtric-docker-net --name=control-panel  nexus3.o-ran-sc.org:10002/o-ran-sc/nonrtric-controlpanel:2.3.0
```

The webbased UI can be accessed by pointing the web-browser to this URL:

http://localhost:8080/

Run the App Catalogue Service Docker Container

Start the App Catalogue Service by the following command.

```
docker run --rm -p 8680:8680 -p 8633:8633 --network=nonrtric-docker-net --name=rapp-catalogue-service nexus3.o-
ran-sc.org:10002/o-ran-sc/nonrtric-plt-rappcatalogue:1.1.0
```

Verify that the service is up and running

```
curl localhost:8680/services
```

Expected output:

```
[ ]
```

Run the Helm Manager Docker Container

Note: Access to kubernetes is required as stated the requirements on the top of this page.

Download the 'helm-manger' repo, Helm Manager.

```
$ cd helmmanager

$ mkdir -p mnt/database

$ kubectl create ns chkm
```

Start the helm manger in a separate shell by the following command:

```
docker run \
    --rm  \
    -it \
    -p 8112:8083  \
    --name helmmanagerservice \
    --network nonrtric-docker-net \
    -v $(pwd)/mnt/database:/var/helm-manager-service \
    -v ~/.kube:/home/nonrtric/.kube \
    -v ~/.helm:/home/nonrtric/.helm \
    -v ~/.config/helm:/home/nonrtric/.config/helm \
    -v ~/.cache/helm:/home/nonrtric/.cache/helm \
    -v $(pwd)/config/application.yaml:/etc/app/helm-manager/application.yaml \
    nexus3.o-ran-sc.org:10002/o-ran-sc/nonrtric-plt-helmmanager:1.2.0
```

Make sure the app has started by listing the current charts - response should be empty json array.

```
$ curl http://helmadmin:itisasecret@localhost:8112/helm/charts
{"charts":[]}
```

To test the app further, start a helm chart repo and create a dummy helm chart

Start a chartmuseum chart repository in a separate shell

```
$ docker run --rm -it \
    -p 8222:8080 \
    --name chartmuseum \
    --network nonrtric-docker-net \
    -e DEBUG=1 \
    -e STORAGE=local \
    -e STORAGE_LOCAL_ROOTDIR=/charts \
    -v $(pwd)/charts:/charts \
    ghcr.io/helm/chartmuseum:v0.13.1
```

Add the chart repo to the helm manager by the following command:

```
$ docker exec -it helmmanagerservice helm repo add cm http://chartmuseum:8080
"cm" has been added to your repositories
```

Create a dummy helm chart for test and package the chart, and save this chart in chartmuserum:

```
$ helm create simple-app
Creating simple-app$ helm package simple-appSuccessfully packaged chart and saved it to: <path-in-current-dir>
/helm-manager/tmp
$ helm package simple-app
Successfully packaged chart and saved it to: <path>/simple-app-0.1.0.tgz

$ curl --data-binary @simple-app-0.1.0.tgz -X POST http://localhost:8222/api/charts
```

The commands below show examples of operations towards the helm manager using the dummy chart.

As an alternative, run the script 'test.sh' to execute a full sequence of commands.

```
$ ./test.sh docker

Start test
================
Get apps - empty
================
curl -sw %{http_code} http://helmadmin:itisasecret@localhost:8112/helm/charts
 Curl OK
  Response: 200
  Body: {"charts":[]}

================
Add repo
================
curl -sw %{http_code} http://helmadmin:itisasecret@localhost:8112/helm/repo -X POST -H Content-Type:application
/json -d @cm-repo.json
 Curl OK
  Response: 201
  Body:

============
Onboard app
===========
curl -sw %{http_code} http://helmadmin:itisasecret@localhost:8112/helm/onboard/chart -X POST -F chart=@simple-
app-0.1.0.tgz -F values=@simple-app-values.yaml -F info=<simple-app.json
 Curl OK
  Response: 200
  Body:

=====================
Get apps - simple-app
=====================
curl -sw %{http_code} http://helmadmin:itisasecret@localhost:8112/helm/charts
 Curl OK
  Response: 200
  Body: {"charts":[{"releaseName":"simpleapp","chartId":{"name":"simple-app","version":"0.1.0"},"namespace":"
ckhm","repository":{"repoName":"cm","protocol":null,"address":null,"port":null,"userName":null,"password":
null},"overrideParams":null}]}

===========
Install app
===========
curl -sw %{http_code} http://helmadmin:itisasecret@localhost:8112/helm/install -X POST -H Content-Type:
application/json -d @simple-app-installation.json
 Curl OK
  Response: 201
  Body:

=====================
Get apps - simple-app
=====================
curl -sw %{http_code} http://helmadmin:itisasecret@localhost:8112/helm/charts
 Curl OK
  Response: 200
  Body: {"charts":[{"releaseName":"simpleapp","chartId":{"name":"simple-app","version":"0.1.0"},"namespace":"
ckhm","repository":{"repoName":"cm","protocol":null,"address":null,"port":null,"userName":null,"password":
```

```
null},"overrideParams":null}]}

================================================================
helm ls to list installed app - simpleapp chart should be visible
================================================================
NAME             NAMESPACE        REVISION        UPDATED
STATUS          CHART                  APP VERSION
simpleapp        ckhm             1               2022-06-27 21:18:27.407666475 +0000 UTC
deployed        simple-app-0.1.0        1.16.0

=========================================
sleep 30 - give the app some time to start
=========================================
============================
List svc and  pod of the app
============================
NAME                   TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)   AGE
simpleapp-simple-app   ClusterIP   10.98.120.189   <none>        80/TCP    30s
NAME                                    READY   STATUS    RESTARTS   AGE
simpleapp-simple-app-675f44fc99-qsxr6   1/1     Running   0          30s

========================
Uninstall app simple-app
========================
curl -sw %{http_code} http://helmadmin:itisasecret@localhost:8112/helm/uninstall/simple-app/0.1.0 -X DELETE
 Curl OK
  Response: 204
  Body:

=========================================
sleep 30 - give the app some time to remove
=========================================
============================================================
List svc and  pod of the app - should be gone or terminating
============================================================
No resources found in ckhm namespace.
No resources found in ckhm namespace.

=====================
Get apps - simple-app
=====================
curl -sw %{http_code} http://helmadmin:itisasecret@localhost:8112/helm/charts
 Curl OK
  Response: 200
  Body: {"charts":[{"releaseName":"simpleapp","chartId":{"name":"simple-app","version":"0.1.0"},"namespace":"
ckhm","repository":{"repoName":"cm","protocol":null,"address":null,"port":null,"userName":null,"password":
null},"overrideParams":null}]}

============
Delete chart
============
curl -sw %{http_code} http://helmadmin:itisasecret@localhost:8112/helm/chart/simple-app/0.1.0 -X DELETE
 Curl OK
  Response: 204
  Body:

================
Get apps - empty
================
curl -sw %{http_code} http://helmadmin:itisasecret@localhost:8112/helm/charts
 Curl OK
  Response: 200
  Body: {"charts":[]}

Test result  All tests ok
End of test
```

To run in the helm manager in kubernetes see this page: Run Helm Manager in kubernetes

Run the Dmaap Adapter Service Docker Container

The Dmaap Adapter Service needs two configurations files, one for the application specific parameters and one for the types the application supports.

Note that a running Information Coordinator Service is needed for creating jobs and a running message router is needed for receiving data that the job can distribute to the consumer.

In addition, if the data is available on a kafka topic then an instance of a running kafka server is needed.

Create the file application.yaml with content below.

The following parameter need to be configured according to hosts and ports (these setting may need to adjusted to your environment)

- ics-base-url: https://information-service-container:8434
- dmaap-base-url: https://message-router:3905 (needed when data is received from the Dmaap message router)
- bootstrap-servers: message-router-kafka:9092 (needed when data is received on a kafka topic)

application.yaml

```
spring:
  profiles:
    active: prod
  main:
    allow-bean-definition-overriding: true
  aop:
    auto: false
management:
  endpoints:
    web:
      exposure:
        # Enabling of springboot actuator features. See springboot documentation.
        include: "loggers,logfile,health,info,metrics,threaddump,heapdump"
springdoc:
  show-actuator: true
logging:
  # Configuration of logging
  level:
    ROOT: ERROR
    org.springframework: ERROR
    org.springframework.data: ERROR
    org.springframework.web.reactive.function.client.ExchangeFunctions: ERROR
    org.oran.dmaapadapter: INFO
  file:
    name: /var/log/dmaap-adapter-service/application.log
server:
   # Configuration of the HTTP/REST server. The parameters are defined and handeled by the springboot framework.
   # See springboot documentation.
   port : 8435
   http-port: 8084
   ssl:
      key-store-type: JKS
      key-store-password: policy_agent
      key-store: /opt/app/dmaap-adapter-service/etc/cert/keystore.jks
      key-password: policy_agent
      key-alias: policy_agent
app:
  webclient:
    # Configuration of the trust store used for the HTTP client (outgoing requests)
    # The file location and the password for the truststore is only relevant if trust-store-used == true
    # Note that the same keystore as for the server is used.
    trust-store-used: false
    trust-store-password: policy_agent
    trust-store: /opt/app/dmaap-adapter-service/etc/cert/truststore.jks
    # Configuration of usage of HTTP Proxy for the southbound accesses.
    # The HTTP proxy (if configured) will only be used for accessing NearRT RIC:s
    http.proxy-host:
    http.proxy-port: 0
  ics-base-url: https://information-service-container:8434
  # Location of the component configuration file. The file will only be used if the Consul database is not used;
  # configuration from the Consul will override the file.
  configuration-filepath: /opt/app/dmaap-adapter-service/data/application_configuration.json
  dmaap-base-url: https://message-router:3905
  # The url used to address this component. This is used as a callback url sent to other components.
  dmaap-adapter-base-url: https://dmaapadapterservice:8435
  # KAFKA boostrap server. This is only needed if there are Information Types that uses a kafkaInputTopic
  kafka:
    bootstrap-servers: message-router-kafka:9092
```

Create the file application_configuration.json according to one of alternatives below.

application_configuration.json without kafka type

application_configuration.json without kafka type

```
{
  "types": [
     {
        "id": "ExampleInformationType",
        "dmaapTopicUrl": "/events/unauthenticated.dmaapadp.json/dmaapadapterproducer/msgs?
timeout=15000&limit=100",
        "useHttpProxy": false
     }
  ]
}
```

application_configuration.json with kafka type

application_configuration.json with kafka type

```
{
  "types": [
     {
        "id": "ExampleInformationType",
        "dmaapTopicUrl": "/events/unauthenticated.dmaapadp.json/dmaapadapterproducer/msgs?
timeout=15000&limit=100",
        "useHttpProxy": false
     },
     {
      "id": "ExampleInformationTypeKafka",
      "kafkaInputTopic": "unauthenticated.dmaapadp_kafka.text",
      "useHttpProxy": false
   }
  ]
}
```

Start the Dmaap Adapter Service in a separate shell with the following command:

```
docker run --rm \
-v <absolute-path-to-config-file>/application.yaml:/opt/app/dmaap-adapter-service/config/application.yaml \
-v <absolute-path-to-config-file>/application_configuration.json:/opt/app/dmaap-adapter-service/data
/application_configuration.json \
-p 9086:8084 -p 9087:8435 --network=nonrtric-docker-net --name=dmaapadapterservice  nexus3.o-ran-sc.org:10002/o-
ran-sc/nonrtric-plt-dmaapadapter:1.1.1
```

Setup jobs to produce data according to the types in application_configuration.json

Create a file job1.json with the job definition (replace paths <url-for-jod-data-delivery> and <url-for-jod-status-delivery> to fit your environment:

job1.json

```
{
  "info_type_id": "ExampleInformationType",
  "job_result_uri": "<url-for-jod-data-delivery>",
  "job_owner": "job1owner",
  "status_notification_uri": "<url-for-jod-status-delivery>",
  "job_definition": {}
}
```

Create job1 for type 'ExampleInformationType'

```
curl -k -X PUT -H Content-Type:application/json https://localhost:8434/data-consumer/v1/info-jobs/job1 --data-
binary @job1.json
```

Check that the job has been enabled - job accepted by the Dmaap Adapter Service

```
curl -k https://localhost:8434/A1-EI/v1/eijobs/job1/status
{"eiJobStatus":"ENABLED"}
```

Data posted on the message router topic unauthenticated.dmaapadp.json will be delivered to the path as specified in the job1.json.

If the kafka type also used, setup a job for that type as well.

Create a file job2.json with the job definition (replace paths <url-for-jod-data-delivery> and <url-for-jod-status-delivery> to fit your enviroment:

job2.json

```
{
  "info_type_id": "ExampleInformationTypeKafka",
  "job_result_uri": "<url-for-jod-data-delivery>",
  "job_owner": "job1owner",
  "status_notification_uri": "<url-for-jod-status-delivery>",
  "job_definition": {}
}
```

Create job2 for type 'ExampleInformationType'

```
curl -k -X PUT -H Content-Type:application/json https://localhost:8434/data-consumer/v1/info-jobs/job2 --data-
binary @job2.json
```

Check that the job has been enabled - job accepted by the Dmaap Adapter Service

```
curl -k https://localhost:8434/A1-EI/v1/eijobs/job2/status
{"eiJobStatus":"ENABLED"}
```

Data posted on the kafka topic unauthenticated.dmaapadp_kafka.text will be delivered to the path as specified in the job2.json.

Run the Dmaap Mediator Producer Docker Container

The Dmaap Mediator Producer needs one configuration file for the types the application supports.

Note that a running Information Coordinator Service is needed for creating jobs and a running message router is needed for receiving data that the job can distribute to the consumer.

Create the file type_config.json with the content below:

type_config.json

```
{
   "types":
     [
       {
         "id": "STD_Fault_Messages",
         "dmaapTopicUrl": "/events/unauthenticated.dmaapmed.json/dmaapmediatorproducer/STD_Fault_Messages?
timeout=15000&limit=100"
       }
   ]
 }
```

There are a number of environment variables that need to be set when starting the application. See these example settings:

- INFO_COORD_ADDR=https://informationservice:8434
- DMAAP_MR_ADDR=https://message-router:390
- LOG_LEVEL=Debug
- INFO_PRODUCER_HOST=https://dmaapmediatorservice
- INFO_PRODUCER_PORT=8185

Start the Dmaap Mediator Producer in a separate shell with the following command:

```
docker run --rm -v \
<absolute-path-to-config-file>/type_config.json:/configs/type_config.json \
-p 8885:8085 -p 8985:8185 --network=nonrtric-docker-net --name=dmaapmediatorservice \
-e "INFO_COORD_ADDR=https://information-service-container:8434" \
-e "DMAAP_MR_ADDR=https://message-router:3905" \
-e "LOG_LEVEL=Debug" \
-e "INFO_PRODUCER_HOST=https://dmaapmediatorservice" \
-e "INFO_PRODUCER_PORT=8185" \
nexus3.o-ran-sc.org:10002/o-ran-sc/nonrtric-plt-dmaapmediatorproducer:1.
1.0
```

Setup jobs to produce data according to the types in type_config.json

Create a file job3.json with the job definition (replace paths <url-for-jod-data-delivery> and <url-for-jod-status-delivery> to fit your environment:

job3.json

```
{
  "info_type_id": "STD_Fault_Messages",
  "job_result_uri": "<url-for-job-data-delivery>",
  "job_owner": "job3owner",
  "status_notification_uri": "<url-for-job-status-delivery>",
  "job_definition": {}
}
```

Create job3 for type 'ExampleInformationType'

```
curl -X PUT -H Content-Type:application/json https://localhost:8083/data-consumer/v1/info-jobs/job3 --data-
binary @job3.json
```

Check that the job has been enabled - job accepted by the Dmaap Adapter Service

```
curl -k https://localhost:8434/A1-EI/v1/eijobs/job3/status
{"eiJobStatus":"ENABLED"}
```

Data posted on the message router topic unauthenticated.dmaapmed.json will be delivered to the path as specified in the job3.json

Run usecases

Within NON-RT RIC a number of usecase implementations are provided. Follow the links below to see how to run them.

- O-RU Fronthaul Recovery usecase
- O-DU Slice Assurance usecase