A4 EMS POD Components Docker Container Installation Guide

December 2018

Revision History

Version	Date	Author	Comments
0.1	December 2018	Updated by Prathap Thammanna and Anand S Katti	Initial Release
0.2	December 2018	Updated by Prathap Thammanna and Anand S Katti	Updated Comments, container boot order and files in appendix with scripts details.
0.3	December 2018	Updated by Prathap Thammanna and Anand S Katti	Update host system details and container boot order sequence update
0.4	December 2018	Updated by Prathap Thammanna and Anand S Katti	Added new section for editing configuration files before installing on new POD. Correct tag name as v0.3 for pod-config and rtb-prometheus-graylog
0.5	December 2018	Updated by Prathap Thammanna and Anand S Katti	Updated tag version for pod-config and olt-control

Contents

Figures

Tables

Chapter 1: Preface

1.1. Objective

The objective of this document is to explain the containerization and launching of dockerized POD services related to EMS workflow. The services for now offer functionality associated with Telemetry and Log collection. The services are realized through open source components and components developed as part of EMS sprints. The details are captured in the following table.

Table 1: Microservices Container Versions

S.NO	Microservices	Type	Version
1	Prometheus	Open Source	2.5.0
2	Alertmanager	Open Source	0.15.3
3	rtb-prometheus-graylog (Graylog Webhook)	Developed in EMS sprints	0.1.1
4	Olt-control (OLT Metric Exporter)	Developed in EMS sprints	0.3
5	Server control(Server Metric Exporter)	Developed in EMS sprints	0.2
6	Graylog	Open Source	2.4
7	Elasticsearch	Open Source	5.6.12
8	Mongo DB	Open Source	3

Prometheus is an open source time series database used for telemetry along with Alertmanager for effective management and handling of alert notifications. Alertmanager routes these notifications to configured end services such as a custom web hook. The web hook handles alert notifications, transforms it and writes to Graylog server as log messages with certain message properties. The OLT-Control and Server-Control modules fetch the telemetry data and expose an exporter for Prometheus to pull the metrics. Applications running inside the containers are here by referred as micro service /service.

Following table provides the host machine details where the docker daemon runs along with all the containers.

Table 2: Host Machine System details

S.No	Host Machine	Version
1	System OS Ubuntu	16.04.X
2	Docker	18.09.0, build 4d60db4
3	Docker-compose	version 1.23.1, build b02f1306

Chapter 2: Container Names

Following are the container names of the micro services.

Table 3: Service and Container Names

Services Name	Container Name
Prometheus	Prometheus
Alertmanager	Alertmanager
rtb-prometheus-graylog (webhook)	rtb-prometheus-graylog
olt-control	olt-control
server-control	server-control
graylog	Graylog
mongo	graylog-mongo
elasticsearch	graylog-elasticsearch

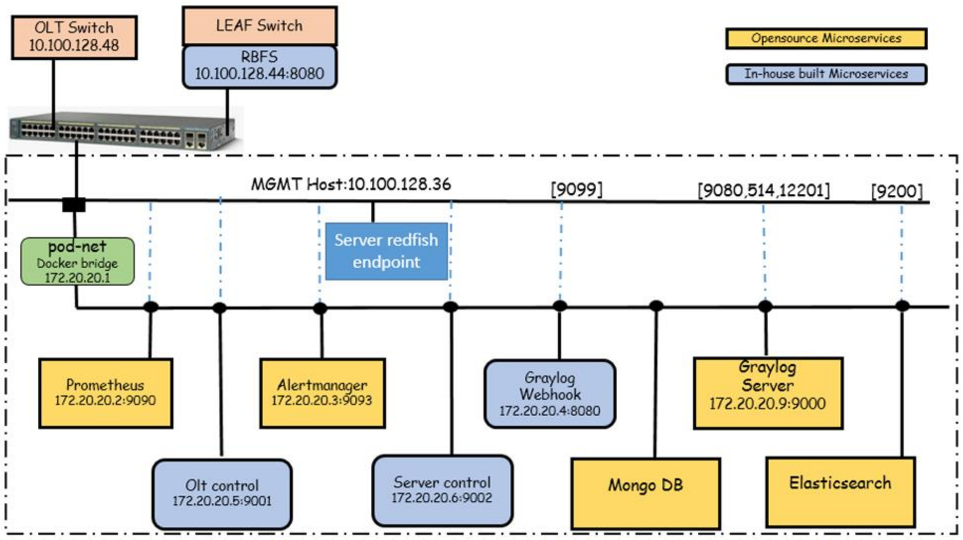
Service names can be used across containers for inter container communication. Docker will resolve the service name to the container endpoints.

2.1. Container Networking

All the containers are interconnected to each other and to the host via Docker Bridge "pod-net". Pod-net bridge is associated with the subnet 172.20.20.0/24 and assigned an IP Address 172.20.20.1. All the containers are given static IP addresses in the same subnet 172.20.20.0/24.

The following Figure 1: Container Networking shows the containers inter networking along with the IP Address of the containers.

Figure 1: Container Networking



OLT Switch 10.100.128.48
LEAF Switch
RBFS 10.100.128.44:8080
Opensource Microservices
In-house built Microservices
MGMT Host:10.100.128.36
[9099]
[9080,514,12201]
[9200]
pod-net Docker bridge 172.20.20.1
Server redfish endpoint
Prometheus 172.20.20.2:9090
Alertmanager 172.20.20.3:9093
Graylog Webhook 172.20.20.4:8080
Graylog Server 172.20.20.9:9000
Olt control 172.20.20.5:9001
Server control 172.20.20.6:9002
Mongo DB
Elasticsearch

2.2. Application Port Mapping

Applications run on the default port inside the containers and they are mapped to ports on the host machine to ease the access of those services from the host machine.

The mapping is described in the following table.

Table 4: Container to Host Ports Mapping

Container Name	Container Port	Host Port
prometheus	9090	9090
alertmanager	9093	9093
rtb-prometheus-graylog (Graylog Webhook)	8080	9099
olt-control	9001	9001
server control	9002	9002
graylog	9000	9080
elasticsearch	9200	9200
mongo	27017	-

2.3. Bindmount - Configuration files and Folders

Each application requires its own configuration files. These configuration files will be required to be updated before running the containers. To avoid repackaging the containers for every modification of the configuration files, these folders are Bindmount to specific folders inside the containers. The local configuration folders on the host machine for each container is as below.

Table 5: Container Bindmounts

Container Name	Host Bindmount folder/file	Path Inside the Container
prometheus	"./prometheus-config/prometheus.yml"	/etc/prometheus/prometheus.yml
prometheus	"./prometheus-config/pod971.rules"	/etc/prometheus/pod971.rules
Alertmanager	"./alertmanager-config/config.yml"	/etc/alertmanager/config.yml
rtb-prometheus-graylog (Graylog Webhook)	"./rtb-prometheus-graylog-config/etc"	/opt/rtbrick/node/rtb-prometheus-graylog-config/etc

2.4. Volume Mounts for Container Storage

Containers which generates a lot of data will consume the storage inside the container and reduce the performance of the container. Thus Volume mounts are defined for storing the container data on to the host machine. Services such as Prometheus and elasticsearch needs storage for storing their time series and log data. Also Alertmanager generates a lot of data for notifications which needs to be stored on the host machine.

Table 6: Container Volume Mounts

Container Name	Volume Mount	Mount Point Inside Container
prometheus	prometheus_data	/prometheus
alertmanager	alertmanager	/alertmanager
graylog	graylog_journal	/usr/share/graylog/data/journal
graylog-elastic	graylog-elastic	/usr/share/elasticsearch/data
graylog-mongo	graylog-mongo	/data/db

Volume mount folders are mapped to specific directories which the micro service application can use it as storage inside the container. The data stored on those folders are actually stored on the host machine in the path **/var/lib/docker/volumes**

```
$docker volume list
```

Shows the details of the volumes created on the host machine.

2.5. Inter-container Communication

There are two mechanisms for Inter container communication, one is by using the statically assigned IP Address as defined in the docker-compose.yml file and other via service names. Docker automatically resolves service names to correct containers. For ex: Alertmanager container can directly write data to graylog webhook by using the service name “rtb-prometheus-graylog”.

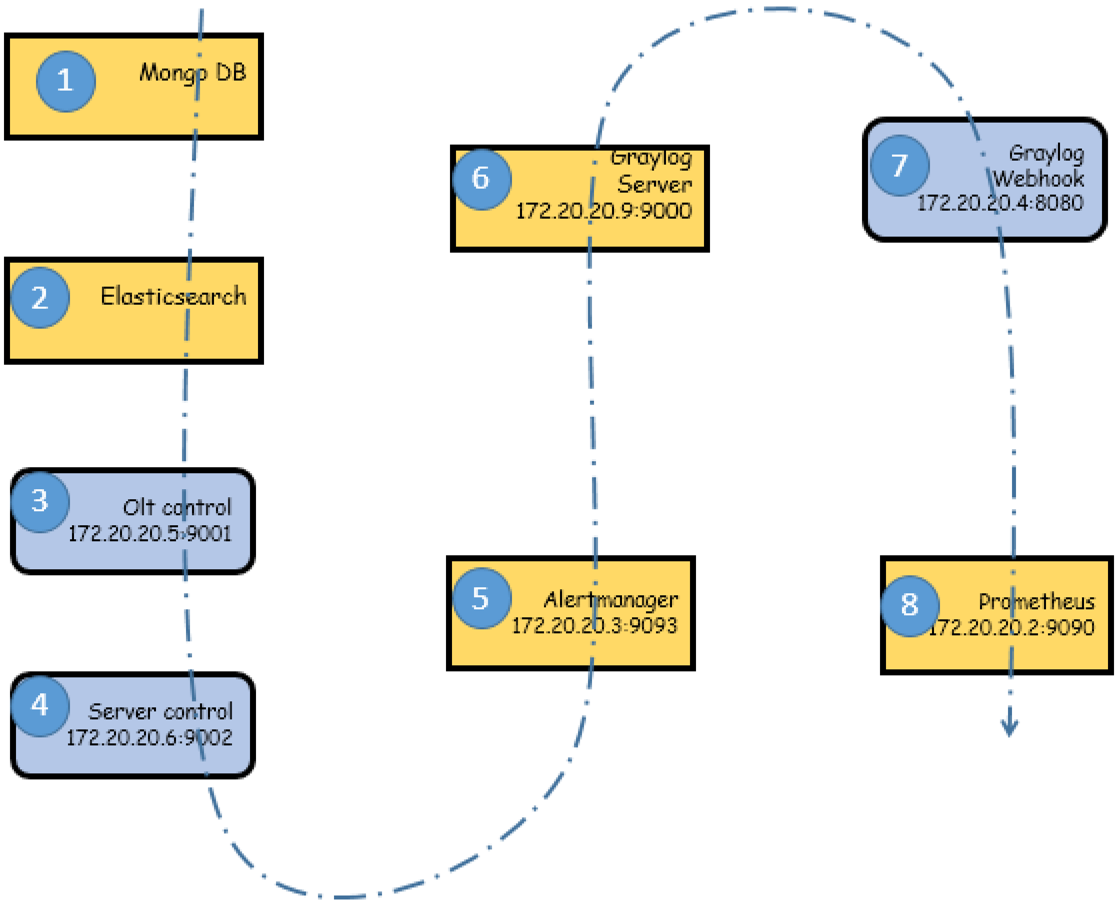
2.6. Container Boot Order

Most of the services we are using have a dependency on one another for proper functioning. For ex: Before the Prometheus and alertmanager starts, the exporters, both olt and server must be up and running. Similarly Graylog, elastic, and mongodb should be up and running before graylog webhook is up.

docker-compose flag “depends_on” is used to ensure containers booting order.

Following block diagram shows the container booting order.

Figure 2: Docker Container Booting Order



Chapter 3: Build/Launch the Containers

3.1. Manage Docker as non-root user

Create docker group

```
$ sudo groupadd docker -f
```

Add your user to the docker group

```
$ sudo usermod -aG docker $USER
```

Log out and log back in so that your group membership is re-evaluated, and then you should be able to issue docker commands as non-root user.

3.2. Install docker-compose

Run following command to install docker-compose.

```
$ sudo curl -L "https://github.com/docker/compose/releases/download/1.23.1/docker-compose-$(uname -s)-$(uname -m)" -o /usr/local/bin/docker-compose

$ sudo chmod  +x /usr/local/bin/docker-compose

$ docker-compose --version

docker-compose version 1.23.1, build b02f1306
```

We use “docker-compose” tool to define multiple containers. These containers are defined in a YAML file which is easy to understand and refer.

Docker-compose provides a lot of utility functions such as build and run the containers, create docker bridges, define and assign container names, create and manage docker volumes and bindmounts and a lot of other useful features.

A sample reference docker compose file is available in Sample docker-compose.yml file in Appendix A A.1

3.3. Build/Pull and save Container Images (Optional)

Skip this section if the container images are already available and placed in certain host path.

3.3.1. Build load and run scripts

Scripts are available in the scripts folder in pod-config repo, to perform the below actions.

Download the pod-config code from the git with relevant tag

```
$ git clone https://gitlab.dol.telekom.de/Access40/pod-config

$ cd pod-config/

$ git fetch -t

$ git checkout tags/v0.3

$ cd scripts
```

Following are the scripts: These scripts can be updated with path, user variables as desired.

```
$ tree scripts/
scripts/
├── git_clone_ems_components.sh
├── load_all_images.sh
├── remove-all-docker-cont-images-vol.sh
├── run-all-docker-containers.sh
└── stop-all-containers.sh
```

1) git_clone_ems_components.sh: This scripts downloads builds and saves the container images in the user defined path in the script

   (default path= /home/$USER/ems_deploy/docker_images/)

2) load_all_images.sh: This script loads all the downloaded container images from the default path, to the local machine docker registry.

   (default path= /home/$USER/ems_deploy/docker_images/)

3) run-all-docker-containers.sh: This script creates containers volumes and networks as defined in the docker-compose.yml file in the pod-config/ directory
4) stop-all-containers.sh: This scripts stops all running containers listed in the docker-compose.yml file
5) remove-all-docker-cont-images-vol.sh: This script removes the container images, volumes and network from the local host as described in the docker-compose.yml

All the exporters, webhook, Prometheus, alertmanager, graylog, elasticsearch and mongodb services needs to be made available as container images.

3.3.2. In-House developed microservices

For in-house built micro services, before building the images, you have to download the specific version /tag of the application.

3.3.2.1. Build and save image of olt-control

To build olt-control container image with source code having tag v0.3

```
$ git clone dev/https://gitlab.dol.telekom.de/Access40/dev/olt-control.git
$ cd olt-control
$ git fetch -t
$ git checkout tags/v0.3
$ docker build . -t olt-control:v0.3
$ docker save -o olt-control_v0.3.tar.gz olt-control:v0.3
```

3.3.2.2. Build and save image of server-control

To build server-control container image having **tag v0.2**

```
$ git clone https://gitlab.dol.telekom.de/Access40/server-control.git
$ cd server-control
$ git fetch -t
$ git checkout tags/v0.2
$ cd server-control/metric_exporter
$ docker build . -t server-control:v0.2
$ docker save -o server-control_v0.2.tar.gz server-control:v0.2
```

3.3.2.3. Build and save image of rtb-prometheus-graylog

Graylog webhook application (rtb-prometheus-graylog) is from RT Brick. It uses the standard prebuilt base image from node and integrates application with it.

For ex: to build graylog webhook, rtb-prometheus-graylog container image having tag v0.1.1

```
$ git clone https://gitlab.dol.telekom.de/Access40/dev/rtb-promtheus-graylog.git
$ cd rtb-prometheus-graylog
$ git fetch -t
$ git checkout tags/v0.2
$ docker build . -t rtb-prometheus-graylog:v0.1.1
$ docker save -o rtb-prometheus-graylog_v0.1.1.tar.gz rtb-prometheus-graylog:v0.1.1
```

3.3.3. Pull and save image of open source containers

These open source applications are available as prebuilt docker images on the internet. Following are the version and steps to pull and save the image tar for further use in the deployments.

3.3.3.1. Pull and save image of Prometheus

Prebuilt docker images for Prometheus is available from the

https://hub.docker.com/r/prom/prometheus

Version 2.5.0 is the latest stable release available on the website.

Run this command to download the prebuilt image on the machine that has internet connectivity.

```
$ docker pull prom/prometheus:v2.5.0
```

Run this command to save the Prometheus image to local machine.

```
$ docker save -o prometheus_v2.5.0.tar prom/prometheus:v2.5.0
```

3.3.3.2. Pull and save image of Alertmanager

Prebuilt docker images for Alertmanager is available from
https://hub.docker.com/r/prom/alertmanager/

Version v0.15.3 is the latest stable release available on the website

Run this command to download the prebuilt image on the machine that has internet connectivity.

```
$ docker pull prom/alertmanager:v0.15.3
```

Run this command to save the Alertmanager image to local machine.

```
$ docker save -o alertmanager_v0.15.3.tar prom/alertmanager:v0.15.3
```

3.3.3.3. Pull and save image of Graylog

Prebuilt docker images for Graylog is available from
https://hub.docker.com/r/graylog/graylog/

Version v2.4 is the latest stable release available on the website

Run this command to download the prebuilt image on the machine that has internet connectivity.

```
$ docker pull graylog/graylog:2.4
```

Run this command to save the Graylog image to local machine.

```
$ docker save -o graylog_v2.4.tar.gz graylog/graylog:2.4
```

3.3.3.4. Pull and save image of Elasticsearch

Prebuilt docker images for Elasticsearch is available from
https://www.elastic.co/guide/en/elasticsearch/reference/5.6/docker.html

Version v5.6.12 is known to work well with graylog 2.4 and is available on the website

Run this command to download the prebuilt image on the machine that has internet connectivity.

```
$ docker pull docker.elastic.co/elasticsearch/elasticsearch:5.6.12
```

Run this command to save the Elasticsearch image to local machine.

```
$ docker save -o elasticsearch_v5.6.12.tar.gz
docker.elastic.co/elasticsearch/elasticsearch:5.6.12
```

3.3.3.5. Pull and save image of Mongo db

Prebuilt docker images for Mongo DB is available from https://hub.docker.com/_/mongo/

Version v3 is known to work well with graylog 2.4 and Elasticsearch v5.6.12 and is available on the website

Run this command to download the prebuilt image on the machine that has internet connectivity.

```
$ docker pull mongo:3
```

Run this command to save the Mongo DB image to local machine.

```
$ docker save -o mongo_v3.0.tar.gz mongo:3
```

3.4. Load and run containers from container images

When the images are made available as tar/tar.gz files and placed in a suitable path in the host machine, use docker load command to load the containers images into the local docker registry. No internet connectivity is required for this option, as the images are already available as container tar/tar.gz files.

For example: if following are the three container images delivered and are available on the MGMT host machine.

```
$ls

rtb-prometheus-graylog.v0.1.tar.gz

prometheus-alertmanager.v0.15.3.tar.gz

prometheus.v2.5.0.tar.gz
```

3.4.1. Load the container images

```
$ docker load -i rtb-prometheus-graylog.v0.1.tar.gz

90d1009ce6fe: Loading layer [==================================================>]
105.5MB/105.5MB

c23711a84ad4: Loading layer [==================================================>]
23.99MB/23.99MB

8f7ee6d76fd9: Loading layer [==================================================>]
8.005MB/8.005MB

f75e64f96dbc: Loading layer [==================================================>]
146.4MB/146.4MB

e02b32b1ff99: Loading layer [==================================================>]
570.6MB/570.6MB

4a6166f16a0e: Loading layer [==================================================>]
349.2kB/349.2kB

ea1edd8bcd15: Loading layer [==================================================>]
62.75MB/62.75MB

c122b6aa3a5a: Loading layer [==================================================>]
5.067MB/5.067MB

6d0553bfb3b4: Loading layer [==================================================>]
3.584kB/3.584kB

799a5dd25997: Loading layer [==================================================>]
4.096kB/4.096kB

737cd128cfc8: Loading layer [==================================================>]
5.12kB/5.12kB

6a5c1fac2493: Loading layer [==================================================>]
8.598MB/8.598MB

d03a6a1c445f: Loading layer [==================================================>]
15.36kB/15.36kB

Loaded image: rtb-prometheus-graylog:latest


$ docker load -i prometheus-alertmanager.v0.15.3.tar.gz
```

```
8a788232037e: Loading layer [==================================================>]
1.37MB/1.37MB

cb53e4a9aceb: Loading layer [==================================================>]
2.618MB/2.618MB

1833c38e6e08: Loading layer [==================================================>]
11.93MB/11.93MB

dabf21ec679d: Loading layer [==================================================>]
20MB/20MB

759f3271c6b1: Loading layer [==================================================>]
3.072kB/3.072kB

Loaded image: prom/alertmanager:v0.15.3


$ docker load -i prometheus.v2.5.0.tar.gz

d3a2b908d5ad: Loading layer [==================================================>]
2.618MB/2.618MB

d30e2865e7f0: Loading layer [==================================================>]
58.08MB/58.08MB

3a5717cf48e2: Loading layer [==================================================>]
39.12MB/39.12MB

441e85dcfd6e: Loading layer [==================================================>]
3.584kB/3.584kB

f38e9cd0b0ae: Loading layer [==================================================>]
12.8kB/12.8kB

14544deca6b4: Loading layer [==================================================>]
28.16kB/28.16kB

d074dfd11ff7: Loading layer [==================================================>]
3.072kB/3.072kB

9b1895a366d3: Loading layer [==================================================>]
5.12kB/5.12kB

Loaded image: prom/prometheus:v2.5.0
```

Similarly load all the remaining container images.

3.4.2. Edit configuration files for installing on new POD

Download the pod-config code from the git with relevant tag

```
$ git clone https://gitlab.dol.telekom.de/Access40/pod-config

$ cd pod-config/

$ git fetch -t

$ git checkout tags/v0.3


Follow the next steps to update the configuration files to match the current POD
settings, then only launch the containers.
```

Based on the details of the new POD, modify the configuration files with information such as pod name, OLT IP Addresses, ILO credentials of server, Leaf switch IP, new prometheus rules file with updated hostnames of olt, server and leaf switch devices etc..

Following are the quick changes to the configuration files (in -config folders) to be done before launching the containers for any new POD

1) In pod-config/prometheus-config/prometheus.yml

   a) Update Leaf Switch exporter IP:

      For job_name: 'rbfs': update the leaf switch IP Address into the - targets field.

      For example: if the target leaf switch IP is 10.100.128.44, then

      Update -targets: ['10.100.128.44:8080'] for job_name 'rbfs' only.

   b) Update rule_files:

      Create a copy of the existing rule file in '. /pod-config/prometheus-config' folder as

      pod971.rules, if the pod id is 971, and update rule_files: with path of the rule file

      For example:

      ```
      rule_files:
          -    "/etc/prometheus/pod971.rules"
      ```

2) In pod-config/prometheus-config/pod971.rules

   a) Update podname field 'pod_name' with pod id

      For example: if the pod id is 971: then pod_name: "POD-971"

   b) Update 'element_name' with hostname for olt, leaf, mgmt host followed by podname.

      For example if the hostname of olt=olt01, server=mgmt01, and leaf=leaf01, then

      i) For alert "OLTTelemetryDataAbsent"

         Update the element name as element_name='olt01.pod971' and update summary field with correct element name => for ex: update summary as "Instance olt01.pod971..."

      ii) For alert "SERVERTelemetryDataAbsent"

         Update the element name as element_name='mgmt01.pod971' and update summary field with correct element_name => for ex: updates summary as

         "Instance mgmt01.pod971..."

      iii) For alert "LEAFTelemetryDataAbsent"

         Update element name as element_name='leaf01.pod971' and update summary field with correct element_name => for ex: updates summary as "Instance leaf01.pod971..."

3) In pod-config/server-control-config/redfish_config.json

   a) Update 'pod_name' with string POD-<PODNUMBER>

For example: if PODNUMBER IS 971, then "pod_name":"POD-971"

b) Update 'element_name' with mgmt hostname and podname.

For example: if mgmt hostname is mgmt01, then element_name="mgmt01.pod971"

i) Update mgmt server ILO IP Address in the 'url'

ii) Update mgmt server ILO credential for 'username' and 'password'

4) In pod-config/olt-control-config/controller_config.yml

a) Update 'deviceips' with any one olt device ip

For example: deviceips: ['10.100.128.48']

b) Update 'podname' with pod id:

For example: podname:"POD-971"

5) In pod-config/docker-compose.yml

a) For service prometheus: update volumes with correct prometheus rule file based on pod number.

For example: for pod971, the bindmount file is pod971.rules and updates it content as mentioned in point 2)

b) Update GRAYLOG_HTTP_EXERNAL_URI with IP of mgmt host on which graylog server is running.

3.4.3. Launch all containers

Run docker-compose command to launch all the containers, it will automatically use the already loaded container images to create containers, and run them with the configuration mentioned in pod-config/docker-compose.yml file and *–config folder

```
$ cd pod-config/scripts
$ docker-compose up –d or ./run-all-docker-containers.sh
```

3.4.4. Create GELF HTTP Input

Run the script graylog_input.py on the HOST machine / VM where all the containers are launched, to create GELF HTTP input in the graylog server

```
$ cd pod-config/scripts
$ python3 graylog_input.py
```

This script waits till the graylog rest interface up, only then it post request to create the GELF HTTP input.

Chapter 4: Commonly Used Commands

Following are some of the commonly performed operations on the containers.

4.1. Login to container

```
$ docker exec –it prometheus sh
$ docker exec –it alertmanager sh
$ docker exec –it rtb-prometheus-graylog bash
```

4.2. Check Container logs:

```
$ docker logs prometheus -f
$ docker logs alertmanager -f
$ docker logs rtb-prometheus-graylog –f
```

4.3. Stop and remove all the containers

Stops containers and removes containers, networks, volumes, and images created by `up`

```
$ docker-compose down -v
```

4.4. Start all the container from a docker compose file

```
$ docker-compose up -d
```

4.5. Usage, logs and list containers

```
$ docker-compose top
$ docker-compose ps
$ docker-compose logs
```

4.6. Check volume details

```
$ docker volume list
$ docker volume inspect <volume name>
```

Appendix A:

Sample Docker compose file defining image and configuration details for containers – Olt Control, Server Control, Prometheus, Alertmanager, Graylog webhook, Graylog Server, Mongo DB and Elasticsearch. Save this compose file as "docker-compose.yml" and use with commands given in this document to launch the containers.

NOTE: Env variable "GRAYLOG_HTTP_EXTERNAL_URI" needs to be updated with the IP Address of the Host Machine where the containers are launched.

Same restart policy of "unless stopped" is used for all other containers.

A.1. Sample docker-compose.yml file

```
version: '3.5'


networks:
  pod-net:
    driver_opts:
      com.docker.network.bridge.name: pod-net
    external: false
    name: pod-net
    ipam:
      config:
        - subnet: 172.20.20.0/24


services:
  prometheus:
    image: prom/prometheus:v2.5.0
    container_name: prometheus
    depends_on:
      - alertmanager
      - olt-control
      - server-control
    ports:
      - 9090:9090
    volumes:
      - "./prometheus-config/prometheus.yml:/etc/prometheus/prometheus.yml"
      - "./prometheus-config/pod928.rules:/etc/prometheus/pod928.rules"
      - "prometheus_data:/prometheus"
    command:
      - '--config.file=/etc/prometheus/prometheus.yml'
      - '--storage.tsdb.path=/prometheus'
    restart: unless-stopped
    networks:
      pod-net:
        ipv4_address: 172.20.20.2

  alertmanager:
    image: prom/alertmanager:v0.15.3
    container_name: alertmanager
    ports:
      - 9093:9093
    volumes:
      - "./alertmanager-config/config.yml:/etc/alertmanager/config.yml"
      - "/alertmanager"
    command:
      - '--config.file=/etc/alertmanager/config.yml'
      - '--storage.path=/alertmanager'
    restart: unless-stopped
    networks:
      pod-net:
        ipv4_address: 172.20.20.3

  rtb-prometheus-graylog:
    image: rtb-prometheus-graylog:v0.1.1
    container_name: rtb-prometheus-graylog
    ports:
      - 9099:8080
    volumes:
      - "./rtb-prometheus-graylog-config/etc:/opt/rtbrick/node/rtb-prometheus-
graylog/etc"
    restart: unless-stopped
    depends_on:
      - graylog
      - olt-control
      - server-control
    networks:
      pod-net:
        ipv4_address: 172.20.20.4

  olt-control:
    image: olt-control:v0.3
    container_name: olt-control
    ports:
      - 9001:9001
    volumes:
      - "./olt-control-config:/home/pod/olt-control-config"
    networks:
      pod-net:
        ipv4_address: 172.20.20.5

  server-control:
    image: server-control:v0.2
    container_name: server-control
    ports:
      - 9002:8000
    volumes:
      - "./server-control-config:/home/pod/server-control-config"
    networks:
      pod-net:
        ipv4_address: 172.20.20.6

  mongo:
    image: mongo:3
    container_name: graylog-mongo
    volumes:
      - "graylog-mongo:/data/db"
    networks:
      pod-net:
        ipv4_address: 172.20.20.7

  # Elasticsearch:
https://www.elastic.co/guide/en/elasticsearch/reference/5.5/docker.html
  elasticsearch:
    image: docker.elastic.co/elasticsearch/elasticsearch:5.6.12
    container_name: graylog-elastic
    environment:
      - http.host=0.0.0.0
      - discovery.type=single-node
      # Disable X-Pack security:
https://www.elastic.co/guide/en/elasticsearch/reference/5.5/security-
settings.html#general-security-settings
      - xpack.security.enabled=false
      - "ES_JAVA_OPTS=-Xms512m -Xmx512m"
    volumes:
      - "graylog-elastic:/usr/share/elasticsearch/data"
    ulimits:
      memlock:
        soft: -1
        hard: -1
    ports:
       9200:9200
      - 9300:9300
    networks:
      pod-net:
          ipv4_address: 172.20.20.8

  # Graylog: https://hub.docker.com/r/graylog/graylog/
  graylog:
    image: graylog/graylog:2.4
    container_name: graylog
    environment:
      - GRAYLOG_PASSWORD_SECRET = somepasswordpepper
      # UserName:Password: admin:admin
      -
GRAYLOG_ROOT_PASSWORD_SHA2=8c6976e5b5410415bde908bd4dee15dfb167a9c873fc4bb8a81f6f2a
b448a918
      - GRAYLOG_HTTP_EXTERNAL_URI=http://172.27.170.167:9080/
      - GRAYLOG_WEB_ENDPOINT_URI=/api
      - GRAYLOG_ENABLE_CORS = true
      - GRAYLOG_ELASTICSEARCH_HOSTS = http://elasticsearch:9200/
      - GRAYLOG_MONGODB_URI = mongodb://mongo/graylog/
      - GRAYLOG_ELASTICSEARCH_REPLICAS = 0
      - GRAYLOG_CONTENT_PACKS_AUTO_LOAD = gelf-http-input.json
      - GRAYLOG_CONTENT_PACKS_LOADER_ENABLED = true
      - GRAYLOG_CONTENT_PACKS_DIR = data/contentpacks
    depends_on:
      - mongo
      - elasticsearch
    volumes:
      - "graylog_journal:/usr/share/graylog/data/journal"
    ports:
      # Graylog web interface and REST API
      - 9080:9000
      # Syslog TCP
      - 514:514
      # Syslog UDP
      - 514:514/udp
      # GELF TCP
      - 12201:12201
      # GELF UDP
      - 12201:12201/udp
    networks:
      pod-net:
        ipv4_address: 172.20.20.9

volumes:
  alertmanager:
    external: false
    name: alertmanager
  graylog-elastic:
    external: false
    name: graylog-elastic
  graylog-mongo:
    external: false
    name: graylog-mongo
  graylog_journal:
    external: false
    name: graylog_journal
  prometheus_data:
    external: false
    name: prometheus_data
```

Appendix B:

B.1. Webhook Dockerfile

'rtb-prometheus-graylog' is the graylog webhook developed by RTB.

Following is the 'Dockerfile' used to build the webhook by pulling the node base image.

To enable dynamic editing of configuration files inside the container, on host machine, folder *./rtb-prometheus-graylog-config/etc* is bind mounted on */opt/rtbrick/node/rtb-prometheus-graylog/etc* inside the container.

```
FROM node

MAINTAINER ems@rtbrick.com

WORKDIR /opt/rtbrick/node/rtb-prometheus-graylog

COPY package*.json ./

RUN npm config set registry https://registry.npmjs.org/

RUN npm install

COPY . .


USER node

CMD ["npm", "start"]
```

B.2. Olt Controller Dockerfile

Olt Control is the OLT metric exporter developed by Radisys.

To enable dynamic editing of configuration files inside the container, on host machine, folder *./olt-control-config*" is bind mounted on */home/pod/olt-control-config*" inside the container.

Restarting container, copies the mounted files

/home/pod/olt-control-config/controller_config.yml to /home/pod/olt-control/conf/ inside the container.

```
FROM ubuntu:16.04

MAINTAINER A4OrgTAC@radisys.com


RUN apt-get update

RUN apt-get install -y python-yaml python3-pip sudo\

     python3-pip python3-dev vim \

      curl sed python-crontab    build-essential libssl-dev libffi-dev python-dev
python3-dev

RUN cd /usr/local/bin \

    && ln -s /usr/bin/python3 python \

    && pip3 install pyyaml  setproctitle setuptools twisted klein python-crontab
paramiko \
    && pip3 install structlog prometheus-client requests datetime progressbar2 \
    && if [ ! -e /usr/bin/pip ]; then ln -s pip3 /usr/bin/pip ; fi \
    && if [[ ! -e /usr/bin/python ]]; then ln -sf /usr/bin/python3 /usr/bin/python;
fi \
    && rm -r /root/.cache

# Copy the current directory contents into the container
ADD . /home/pod/olt-control/.

# Create pod user and group
RUN useradd -m pod -s /bin/bash && echo "pod:pod" | chpasswd && adduser pod sudo

RUN chown -R pod:pod /home/pod/olt-control/ && \
    chmod -R 777 /home/pod/olt-control

WORKDIR /home/pod/olt-control

RUN echo "pod ALL=(root) NOPASSWD:ALL" > /etc/sudoers.d/pod && \
    chmod 0440 /etc/sudoers.d/pod

RUN sudo mkdir -p /var/log/a4/
USER pod

ENTRYPOINT ["/bin/bash", "./run.sh"]
```

B.2.1. run.sh startup script for Olt Controller container

```
#!/bin/bash

# Check for the file
CONFIG="/home/pod/olt-control-config/controller_config.yml"
if [ -f "$CONFIG" ];
then
     echo "$CONFIG found."
     cp  $CONFIG conf/.
else
     echo -e "$CONFIG File is not there"
fi

./start_device_exporter.sh
```

B.2.2. Server Controller Dockerfile

```
#Base image python:3
FROM python:3

#Add current directory in container into "/exporter"
ADD / /exporter

#Set /exporter/python as working directory
WORKDIR /exporter/python

#Add user for running the Exporter
RUN groupadd -g 999 pod  && \
    useradd -r -u 999 -g pod pod

#Preparation of logging directory
RUN mkdir /var/log/a4 && \
    chown -R :pod /var/log/a4 && \
    chmod -R g+rw /var/log/a4

#Grant read and write permission to pod-users for /exporter
RUN chown -R :pod /exporter/ && \
    chmod -R g+rw /exporter/

#Install nano to change configurations interactive
RUN apt-get update && \
    apt-get install -y nano

#Install required python modules
RUN pip3 install pystache \
                 structlog \
                 requests \
                 nested-lookup

#Define user "pod"
USER pod

#Run exporter script
```

```
CMD ["../docker-run.sh"]
```

B.2.2.1. Docker-run.sh startup script for Server Controller

```
#!/bin/bash

#Starting script for docker container to start Exporter
#Author: Cedric Spengler, cedric-spengler@telekom.de
#Copyright: (c) Copyright 2017-18 Deutsche Telekom Technik GmbH

SERVER_CONFIG_PATH=/home/pod/server-control-config/server_config.json
REDFISH_CONFIG_PATH=/home/pod/server-control-config/redfish_config.json

if [[ -f $SERVER_CONFIG_PATH ]]; then
   cp $SERVER_CONFIG_PATH /exporter/config/
fi

if [[ -f $REDFISH_CONFIG_PATH ]]; then
   cp $REDFISH_CONFIG_PATH /exporter/config/
fi

python3 -u prometheus_exporter.py -L /var/log/a4
```

Appendix C:

C.1. POD-Config repo

Folder names ending with **"–config"** are config directories for each micro service with their configuration file. These –config folders are bind mounted into the micro service container, so that dynamically configuration file can be edited without having to rebuild a new container image.

```
$ tree pod-config/
pod-config/
├── alertmanager-config
│   └── config.yml
├── docker-compose.yml
├── olt-control-config
│     └── controller_config.yml
│
├── prometheus-config
│   ├── pod928.rules
│   └── prometheus.yml
├── README.md
├── rtb-prometheus-graylog-config
│   └── etc
│       └── connector.yaml
├── scripts
│   ├── git_clone_ems_components.sh
│   ├── load_all.sh
│   ├── remove-all-docker-cont-images-vol.sh
│   ├── run-all-docker-containers.sh
│   ├── stop-all-containers.sh
    └── graylog_input.py
└── server-control-config
    ├── redfish_config.json
    └── server_config.json

7 directories, 15 files
```