OMP MENTORSHIP PROGRAM 2020: PROJECT ZEBRA

Project Team
Yongkook(Alex) Kim
Salisu Ali

For:
Zebra V1.0.0

Contents:

User Documentation

Contributor Documentation

OMP Mentorship Program 2020: Project Zebra

Setting up your Environment

In order to run zebra, you will need to set up your environment by installing Nodejs version 8.11.2. Installing Nodejs comes along with npm. Git is also needed for cloning the app from a github repository. Prometheus, MongoDB and Grafana are other programs needed to run Zebra.

Clone Zebra

To clone zebra from OMP github repository, use the command:

- o git clone https://github.com/openmainframeproject-internship/Zowe-Parsing-Engine-for-SMF-or-RMF-PP-Reports.git

Installing App Packages

Follow these steps to install the app packages using a terminal/command prompt:

- cd into the cloned app src folder
- run npm install.

The app will install packages as contained in the package.json folder.

How to Run App

To run Zebra App on a server or Local Machine, a user can choose to run the app using npm, nodemon or pm2.

i.   **Using NPM (For Testing/Contributing)**
     Follow these steps to run Zebra App using npm from a terminal/command prompt:
- o cd into the cloned app src folder (if you are not there already).
- o run npm start

Note: using this method, a user has to stop and restart zebra whenever he/she made a change to the apps configuration for the changes to take effect.

ii. **Using Nodemon (For Testing/Contributing)**
     After installing nodemon, follow these steps to run Zebra App using nodemon from a terminal/command prompt:
- o cd into the cloned app src folder (if you are not there already).
- o run nodemon

Note: using this method, Zebra Automatically restart whenever a user made a change to the apps configuration.

iii.   Using PM2

Follow these steps to run Zebra App using pm2 from a terminal/command prompt:
- cd into the cloned app src folder (if you are not there already).
- run pm2 start ./bin/www

Note: use this method for production, it helps manage and keep your application running 24/7

The App User Interface:

The Homepage

The homepage contains sections for trying the API request from Zebra by simply inputting the report title in a textbox and a section to connect Zebra to MongoDB and Grafana.

Click on try it to request data from RMF in JSON Format

Click on Browse RMF data from MongoDB to connect to MongoDB

Click on Browse RMF real-time data with Grafana to connect to Grafana
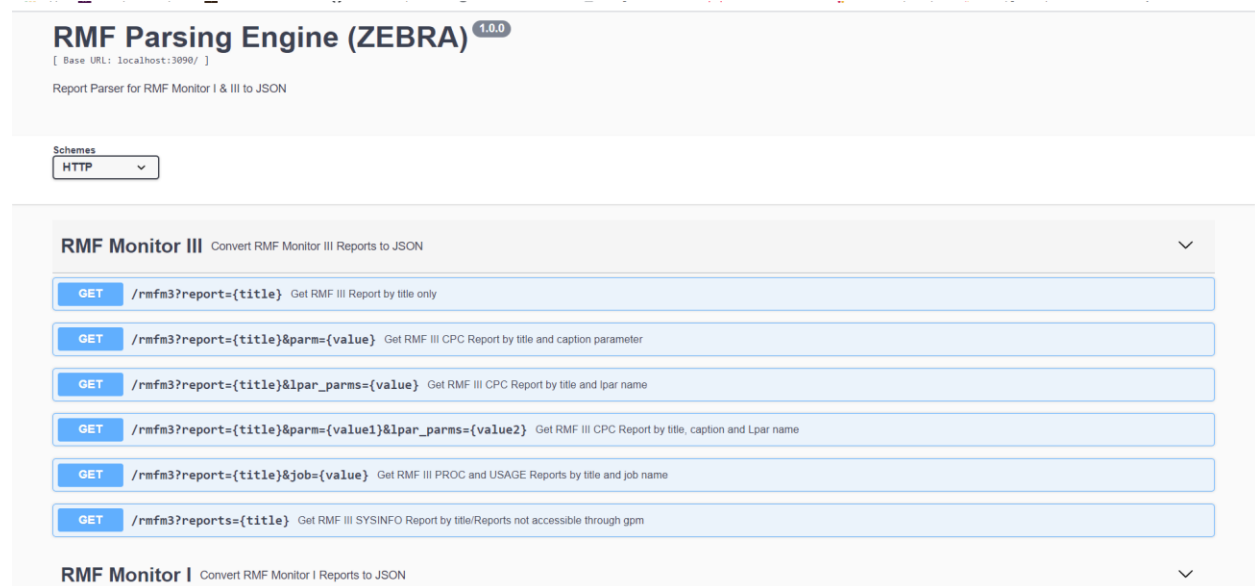
All API's

This page displays a swagger documentation of all ZEBRA API's for RMF Monitor III, RMF Monitor I and The Static RMF xml file. A user can try to access data using this swagger page.

Click on GET to try the API

Manage Files

This page allows users to parse new RMF xml files and view their recent files, as well as delete those recent files.

Click on Choose file to select a static RMF xml file on Disk

Click on Parse to parse a file to JSON

Click on Delete to delete the file

Mongo

This Page allows for retrieving MongoDB data.

- Report type takes the Report title of retrieve(Mandatory)

- Report date takes the date of the reports to retrieve. Its in the format MM/DD/YYYY. (Optional)
- Report Duration takes the time range of the reports to retrieve. It requires Report date to be specified. It's in the format starttime:stoptime. E.g. 06.30.00:07.50.56 (Optional but Requires Report date)
- Report time takes the value of a specific time of a report to retrieve. The time format is HH.MM.SS E.g 05.31.30. If report is not available at this exact time, a close report to this time is displayed. (Optional but Requires Report date)

Click Get Data to retrieve data from MongoDB.

Configure App

Zebra's configuration gives users the flexibility to run the app according to their needs. Zebra has its Recognized configuration parameters as follows:

Recognized Zebra Config Parameters

- **ddsbaseurl: (Distributed Data Server base URL)**
  This is the IP address of the DDS from which the App can retrieve RMF Data. E.g.

  ddsurl: 127.0.0.1
- **ddsbaseport: (Distributed Data Server base Port)**
  This is the Port number of the DDS from which from which the App can retrieve RMF Data. E.g

  ddsport: 8888

- **rmf3filename: (DDS RMF Monitor III Filename)**
  This is the filename from which The App can retrieve RMF Monitor III data. E.g.

  rmf3filename: rmfm3.xml

- **rmfppfilename**: **(DDS RMF Monitor I Filename)**
  This is the filename from which The App can retrieve RMF Monitor I data. E.g.

  rmfppfilename: rmfpp.xml

- **mvsResource: (DDS RMF Monitor III resource identifier)**
  This parameter represent Monitor III resource identifier. E.g.

  mvsResource: ,SYS,RESOURCE

- **mongourl: (Mongo DB URL)**
  This is the IP address of MongoDB to which RMF III data will be saved. E.g.

  mongourl: 127.0.0.1

- **dbinterval: (Database Interval)**
  This is the Interval for which data will be saved to MongoDB. Its unit is in seconds. E.g. Every 100 seconds

  dbinterval: 100

- **dbname: (Database Name)**
  This is the name of the database for which RMF III data will be saved. E.g

  dbname: ompzebra

- **appurl: (Running Zebra App URL)**
  This is the IP address of running Zebra instance (after hosting). This value is needed for MongoDB, Prometheus and Grafana to work with Zebra. E.g. 127.0.0.1.

  appurl: 127.0.0.1

- **appport: (Running Zebra App Port)**
  This is the Port of a running Zebra Instance (after hosting). This value is needed for MongoDB, Prometheus and Grafana to work with Zebra. E.g. 3000

  appport: 3000

- **mongoport: (Mongo DB Port)**
  This is the port number of MongoDB to which RMF III data will be saved. E.g.

  mongoport: 27017

- **ppminutesInterval: (RMF Monitor I PP Report Interval)**
  This is the Interval for which DDS Produce RMF I report. Its unit is in minutes
  E.g. Every 30 Minutes

<div align="center">

ppminutesInterval: 30

</div>

- **rmf3interval: (RMF Monitor III Report Interval)**
  This is the Interval for which DDS Produce RMF III report. E.g. Every 100
  seconds

<div align="center">

rmf3interval: 100

</div>

- **httptype: (http type of running Zebra App)**
  This is the hypertext transfer protocol type of the running zebra app (after
  hosting). Its value is either http or https. E.g.

<div align="center">

httptype: http

</div>

- **useDbAuth: (Use Database Authentication)**
  This config parameter determines the type of connection to mongodb, either with
  authentication or without authentication. Its value is either **true** or **false**. If value is
  set to true, zebra will require username and password with which to connect to
  mongodb. E.g.

<div align="center">

useDbAuth: true

</div>

- **dbUser: (Database Username)**
  This is mongodb username with which zebra will connect to database if value of
  useDbAuth is set to true. E.g.

<div align="center">

dbUser: salisuali

</div>

- **dbPassword: (Database Password)**
  This is mongodb password with which zebra will connect to database if value of
  useDbAuth is set to true. E.g.

<div align="center">

dbPassword: salisu

</div>

- **authSource: (Authentication Source)**
  This is mongodb database which contains the username and password for
  authentication. The default is mongodb's "**admin**" database. E.g.

<div align="center">

authSource: admin

</div>

- **useMongo: (Use Mongo DB)**
  This parameter needs to be set to true before a user can connect MongoDB to
  Zebra. Its default value is false. E.g.

<div align="center">useMongo: true</div>

- **usePrometheus: (Use Prometheus server)**
  This parameter needs to be set to true before a user can connect Prometheus to Zebra. Its default value is false. E.g.

<div align="center">usePrometheus: true</div>

- **https: (Use TLS)**
  This parameter needs to be set to true before a user can use TLS for Zebra. Its default value is false. E.g.

<div align="center">https: true</div>

- <u>**grafanaurl:**</u>
  This is the IP address of running Grafana instance. E.g. 127.0.0.1.

<div align="center"><u>grafanaurl: 127.0.0.1</u></div>

- <u>**grafanaport:**</u>
  This is the Port of a running Grafana instance. E.g. 3000

<div align="center"><u>grafanaport: 3000</u></div>

HOW To Configure APP

Users can configure Zebra by

1. Editing the Zconfig.json File in an Editor/IDE.
2. **<u>Using Zebra UI</u>**
   i. Click on Settings from the Homepage Navigation Bar
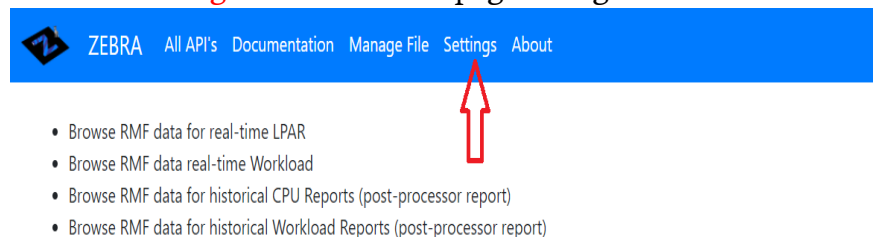
   ii. If you are not logged in, you will be redirected to the login page.
   Zebra comes with one default Admin account. The default username/Password is Admin/Admin. Zebra will prompt the user to change the password of this account before first use.

If login for the first time, the user will be prompted to change your password. No change needed for the username.

After successfully changing the password. The user is logged into the App as Admin and redirected to the homepage. The user will need to click on Settings again from the Navigation Bar. This time, no login will be required.

iii.   After clicking on the Settings page, A refresh token, Access token and parameters from the Zconfig file will be displayed to the user:

Access token last for 15 Minutes, In case of an expired Access token, A User can click on New Access Token Button. This automatically generates a new Access token which a User can use to change the configurations of the App.

After changing the required Zconfig parameter values. A user can click on save button to save this changes to the app.

iv. Likely Messages
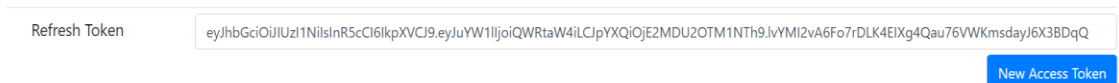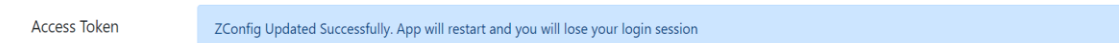
Solution: Click New Access Token Button

Update successful..

3. **By Sending a POST request to Zebra's /addsettings Endpoint.**

Zebra provides user Authentication through Java/JSON web Token (JWT). To access to Of Zebra's Endpoint (**/settings** and **/addsettings**). A user needs to have a username and password for login. Access and refresh tokens are generated after a successful login. Access Tokens will be needed to send request to Zebra's Endpoints Mentioned above while the Refresh token will be needed to generate new access tokens. At the moment, access tokens expire every 15 minutes.

In this guide, I made use of VS code Rest Client Extention to make Http request to Zebra. However, one can choose to use other programs like postman, curl etc
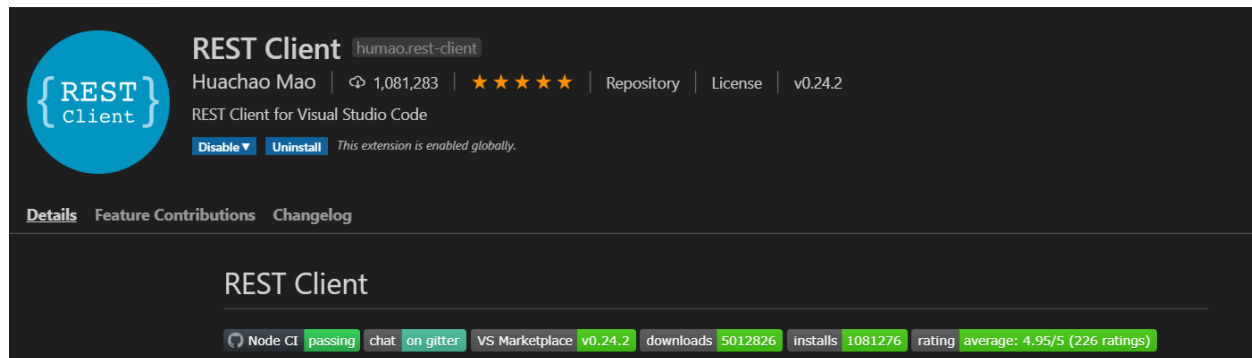
HOW TO LOG IN

Zebra provides a default Admin username and password as **Admin/Admin**. Zebra also provide the **/login** Endpoint through which a user can send a **POST** Request. The header content type is JSON and the keys of the JSON are:

- name: this takes the value of the username
- password: this takes the value of the password

```
Send Request
1  POST http://localhost:3090/login
2  Content-Type: application/json
3
4  {
5      "name": "Admin",
6      "password": "Admin"
7  }
8
```

If POST request is Successful, Zebra Returns a welcome message, Access token and refresh token

```
1    HTTP/1.1 200 OK
2    X-Powered-By: Express
3    Content-Type: text/html; charset=utf-8
4    Content-Length: 317
5    ETag: W/"13d-iaD4kg0ZWvw1eYatTCo+V2I5q4M"
6    Date: Tue, 08 Sep 2020 22:29:16 GMT
7    Connection: close
8
9    Welcome Admin
10   Access Token: eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJuYW1lIjoiQWRtaW4iLCJpYXQiOjE1Tk2MDQx
     NTYsImV4cCI6MTU5OTYwNDIxNn0.7_cDfxLYdU4Ua08mLjZ8z2nvquEwPQQ7COLx63gGBRE
11   Refresh Token: eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJuYW1lIjoiQWRtaW4iLCJpYXQiOjE1Tk2MDQ
     xNTZ9.Whxrqb3m2Be6yXRwDHW_R9HCwnXQlqxhoPp8KJD0m_Q
```

HOW TO UPDATE PASSWORD

   Zebra Admins are encouraged to Change this password. After a Successful login, Admins can use their access token to change the default password associated with the admin account.

   This also Involves sending a **POST** request to Zebra's **/UpdatePassword** Endpoint. The header of this request Contains Authorization parameter whose value is **Bearer <Access token>**. The Content body contains two keys:

- **oldpassword**: whose value is the password to change
- **newpassword:** whose value is the new password to associate with the admin account

```
9    ###
     Send Request
10   POST http://localhost:3090/UpdatePassword
11   Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJuYW1lIjoiQWRtaW
12   Content-Type: application/json
13
14   {
15       "oldpassword": "Admin",
16       "newpassword": "Zebra"
17   }
18
```

If POST request is Successful, the user gets a message saying "**Password Updated Successfully**"

```
1    HTTP/1.1 200 OK
2    X-Powered-By: Express
3    Content-Type: text/html; charset=utf-8
4    Content-Length: 29
5    ETag: W/"1d-9+RMiGH2WYM5pBbjBejllTWRfpA"
6    Date: Tue, 08 Sep 2020 22:35:22 GMT
7    Connection: close
8
9    Password Updated Successfully
```

HOW TO GENERATE NEW ACCESS TOKEN

Access tokens last for 15 minute. To generate new Access token, Zebra Provides the **/token** Endpoint. This involves sending a **POST** request with the users refresh token in the request body. The request body has one key:

- **token**: the value of this key is the refresh token enclosed in a quotation mark.

```
38    ###
      Send Request
39    POST http://localhost:3090/token
40    Content-Type: application/json
41
42    {
43        "token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJuYW1lIjoiQWR
44    }
45
```

If Request is successful, Zebra returns a new access token for the user.

```
 1   HTTP/1.1 200 OK
 2   X-Powered-By: Express
 3   Content-Type: application/json; charset=utf-8
 4   Content-Length: 166
 5   ETag: W/"a6-7cDIGEyiIzCuD79/JfBuhBdDtas"
 6   Date: Tue, 08 Sep 2020 22:47:48 GMT
 7   Connection: close
 8
 9 ∨ {
10     "accessToken": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJuYW1lIjoiQWRtaW4iLCJpYXQiOjI
       MDUyNjgsImV4cCI6MTU5OTYwNTMyOH0.edWcMh1mpBNVVQ10URUxvr3K8wqWg83xQqbHkKQSozE"
11   }
```

ACCESSING ZEBRA'S SECURE ENDPOINTS

  A user can send a **GET** request to Zebra's **/settings** Endpoint to view his current Configuration. This requires only the Authorization parameter containing a valid access token in the request header.

In the case of changing Zebra's Configurations, the user needs to send a **POST** request to **/addsettings** with both Authorization and contents as part of the request header. The Authorization parameter should contain a valid access token while the body of the content can contain any of zebra's recognized parameter and its value (Multiple parameters are acceptable).

```
23   ###
     Send Request
24   GET http://localhost:3090/settings
25   Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJuYW1lIjoiQWRtaW4iLCJp
26
27   ###
     Send Request
28   POST http://localhost:3090/addsettings
29   Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJuYW1lIjoiQWRtaW4iLCJp
30   Content-Type: application/json
31
32   {
33       "ddsurl":"salisuali.com",
34       "ddsport":"5906",
35       "usePrometheus": "false"
36   }
37
```

LOGOUT

Refresh tokens don't have an expiration time but the user can choose to delete his refresh token by send a **POST** request to Zebra's **/logout** Endpoint. This request requires the Authorization parameter with a valid Access token.

```
46    ###
      Send Request
47    GET http://localhost:3090/logout
48    Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJu
49
```

If The Request is Successful, Zebra returns a message "**Done**" and the refresh token has been deleted. To get a new refresh token, a user has to login again.

```
1    HTTP/1.1 200 OK
2    X-Powered-By: Express
3    Content-Type: text/html; charset=utf-8
4    Content-Length: 8
5    ETag: W/"8-RHy4RbDleIIaPG8AkerNyWSN0xU"
6    Date: Tue, 08 Sep 2020 23:03:58 GMT
7    Connection: close
8
9    Done!!!!
```

Note: Replace localhost:3090 with the IP and port of your running Zebra Instance

How to Set Up HTTPS for Zebra

To set up TLS for Zebra Using your own CA certificate and key, follow these steps:

1. **Edit server.cert and server.key**

   In the cloned src folder of Zebra, a subfolder named sslcert contains the files needed to set up TLS for zebra.

   i.   **Server.cert**

        This file contains the Certificate to use in setting up TLS. Edit it by deleting its content and adding your own certificate.

   ii. **Server.key**

        This file contains the private key to use in setting up TLS. Edit it by deleting its content and adding your certificate's private Key.

2. **Edit Zconfig**

After adding your private key and certificate to zebra, You will need to edit 3 Zebra config parameters in the Zconfig.json file.

i. **https**

This parameter needs to be set to **true** for zebra to use TLS.

ii. **appport**

The default port for zebra is 3090. Zebra makes use of port 3090 for http connections. In case of setting up TLS for Zebra, a different port is is used by zebra. The new port is calculated using the formula

$$\text{TLS port} = (\text{Default port} + 1)$$

The port for TLS connection is now 3091. So, appport parameter needs to be set to 3091.

If you decided to change your default port for zebra, you can also calculate your https port value and set appport to the value of your TLS port.

iii. **httptype**

Finally, you will need to change this parameters value to https

Note: if you are running Zebra using management tool like npm (which does not check for file changes). You will need to restart zebra for changes to take effect. Default Port (3090) automatically stops working after setting up TLS.
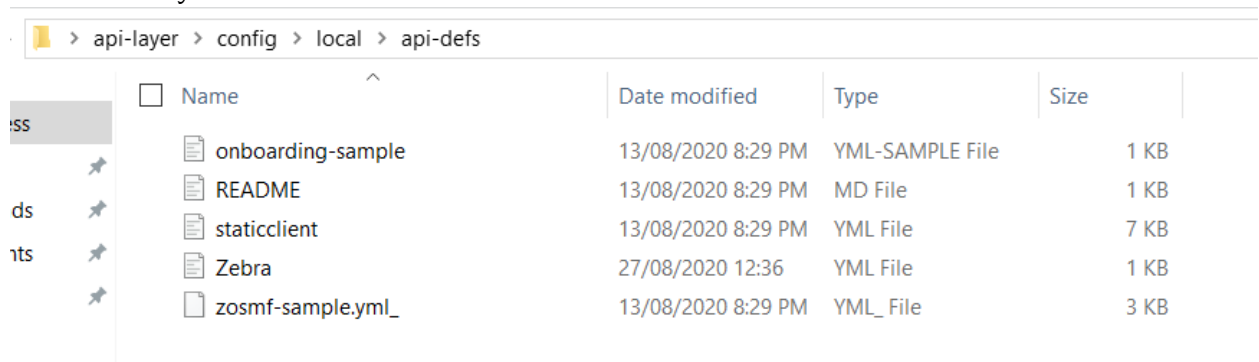
How To Add Zebra To Zowe API Mediation Layer

To add Zebra to API ML. Follow these steps:

1. **Copy Zebra.yml to API ML**

From the src folder of cloned Zebra, Edit the Zebra.yml by changing the value of swaggerUrl to the url of your running Zebra instance.

Then Copy the Zebra.yml file to Zowe API Mediation Layer's config/local/api-defs directory.

	Name	Date modified	Type	Size
	onboarding-sample	13/08/2020 8:29 PM	YML-SAMPLE File	1 KB
	README	13/08/2020 8:29 PM	MD File	1 KB
	staticclient	13/08/2020 8:29 PM	YML File	7 KB
	Zebra	27/08/2020 12:36	YML File	1 KB
	zosmf-sample.yml_	13/08/2020 8:29 PM	YML_ File	3 KB

> api-layer > config > local > api-defs

2. **Edit Zebra_swagger.json**
   In the Zebra_Swagger.json file, you will need to edit the host and schemes parameters with the correct values of your running zebra instance.

How to Connect MongoDB to Zebra

The first step in connecting MongoDB to zebra is setting the value of **useMongo** config parameter to "**true**".

i.   **Without Authentication**
      If a user choose to connect Zebra to MongoDB without Authentication, he/she will need to set **useDbAuth** parameter to **false** and provide the correct values for other config parameters.

ii.   **With Authentication**
      If a user choose to connect to MongoDB using username and password. The user will need to follow these procedures:

   a. **Enable MongoDB Access Control**
      To Enable MongoDB Access Control, the user will first need to create a Username and password with a read Write Permission/Role. To do this, the user will need to:

   o **Connect to the MongoDB instance**
      For example, open a new terminal and connect a mongo shell to the instance:

```
mongo --port 27017
```

      Specify additional command line options as appropriate to connect the mongo shell to your deployment, such as –**host**

   o **Create the user administrator**
      From the mongo shell, add a user with the **userAdminAnyDatabase** role in the admin database. Include additional roles as needed for this user. For example, the following creates the user **myUserAdmin** in the admin database with the **userAdminAnyDatabase** role and the **readWriteAnyDatabase** role.

```
use admin
db.createUser(
  {
    user: "myUserAdmin",
    pwd: passwordPrompt(), // or cleartext password
    roles: [ { role: "userAdminAnyDatabase", db: "admin" }, "readWriteAnyDatabase" ]
  }
)
```

b. Modify MongoDB Configuration File

The next step is to add the **security.authorization** configuration file setting:

```
security:
     authorization: enabled
```

Note: after making the above changes, **Re-start the MongoDB instance**

c. Change Zebra Config Parameters

To use MongoDB with Authentication from Zebra, a user need to set **useDbAuth** parameter to **true**. The user should also provide values for **dbUser**, **dbPassword**, **authSource** and the correct values for remaining config parameters.

How to Connect Prometheus to Zebra

The first step in connecting MongoDB to zebra is setting the value of **usePrometheus** config parameter to "**true**".

Zebra provides "**/prommetric**" Endpoint that expose Custom Prometheus metrics. Users can scrape these metrics by using Prometheus. Simply modify the config file of Prometheus to point to Zebra's "**/prommetric**" Endpoint:

Note: Modify the value for target to point to IP address and port of your running Zebra Instance.

How to Connect Grafana to Prometheus

When Prometheus has been connected to Zebra, the values scraped by Prometheus can be used to create Realtime dashboards using Grafana.

- **Data Source**

    Prometheus will be our data source and the data it scrapes will be used to plot our real time chart. Use the following steps to add Prometheus as a data source to Grafana.
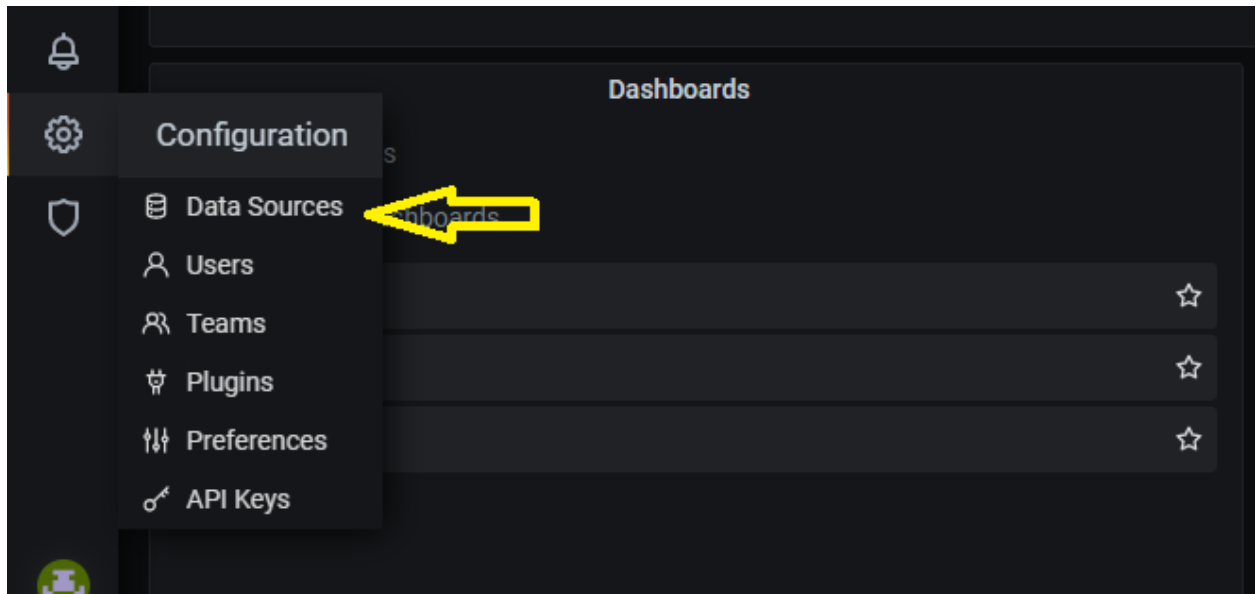
   i.   After logging into Grafana, click on the **configuration button** and select **Data source** (as shown by the yellow arrow in the image above).

   ii.   Click on **Add data source** button (as shown by the yellow arrow in the image below)
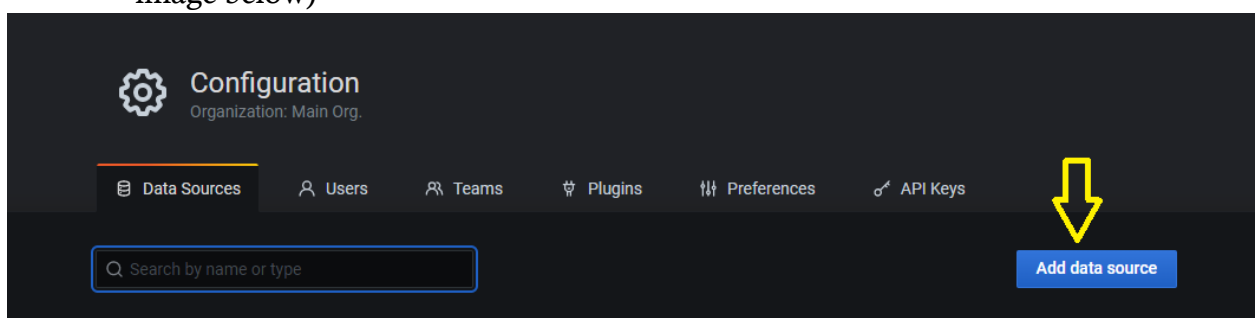
   iii.   Hover over Prometheus option and click on **select** (as shown by the yellow arrow in the image below)
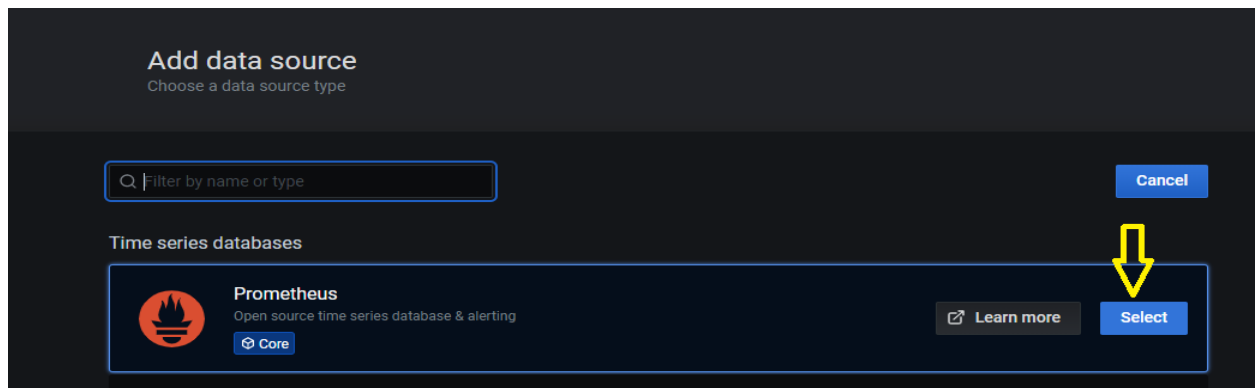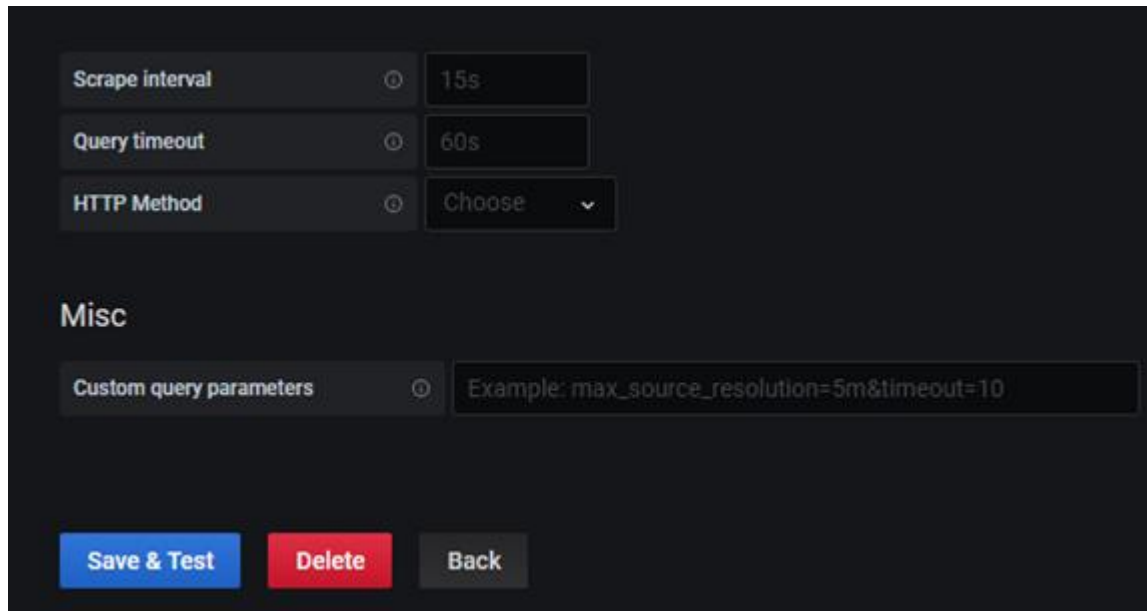
iv.   Choose a **name** for the data source and enter the **URL** of your running
        Prometheus instance.
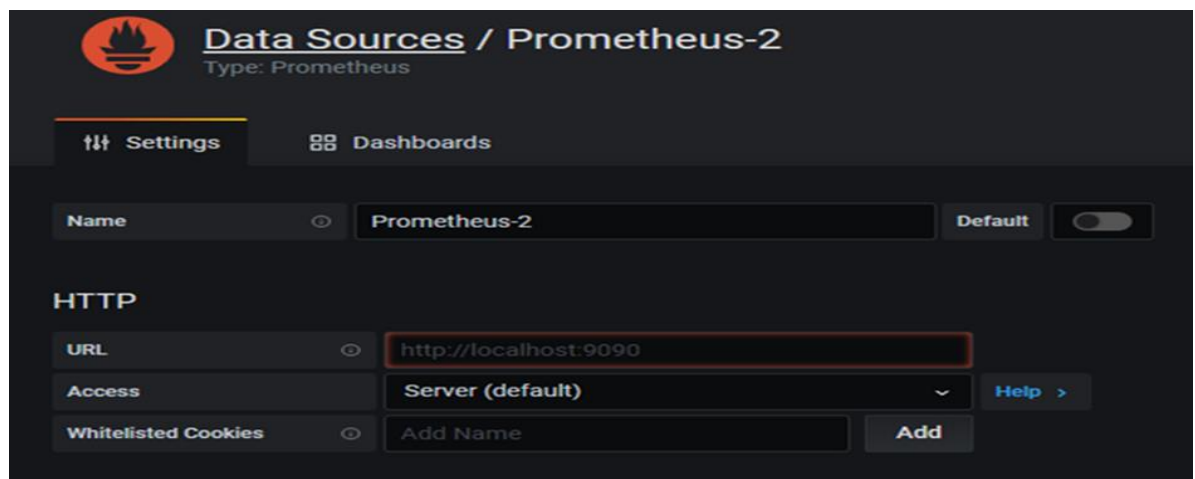
v.   You can choose to change the values for **scrape interval, Query timeout
        and HTTP method**. You can as well choose to stick with the default
        values.

Finally, Click on **Save & Test** button as seen in the image above.

- **Build You own Dashboard**

    After successfully adding Prometheus as data source to Grafana. You can now create your real time dashboards. Use the following steps to create your dashboard:

    i-   From Grafana homepage, click on create "+" button (as shown by the yellow arrow in the image below)
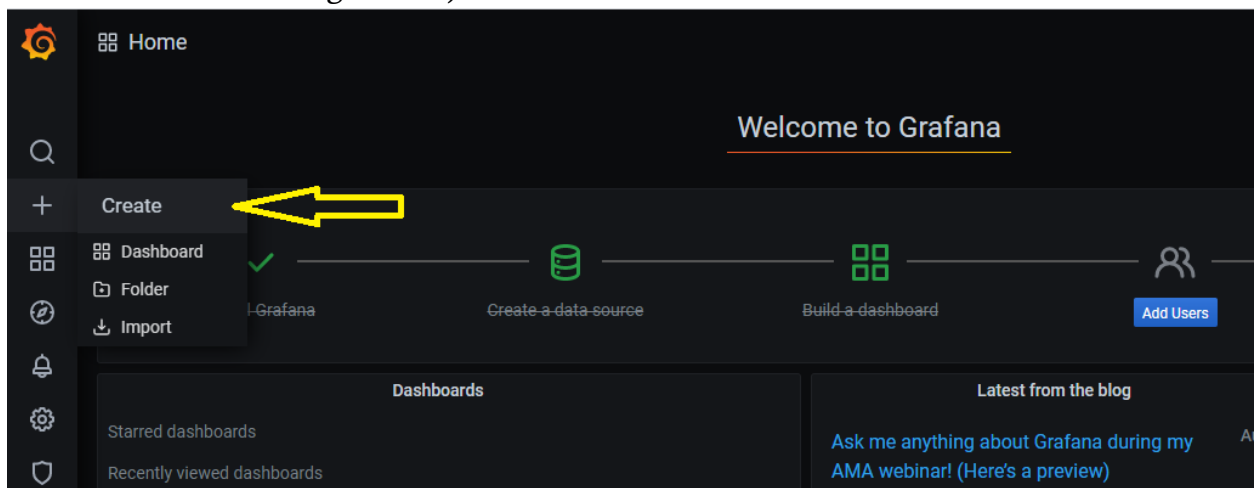
    ii-   Click on Add new panel button (as shown by the yellow arrow in the image below)
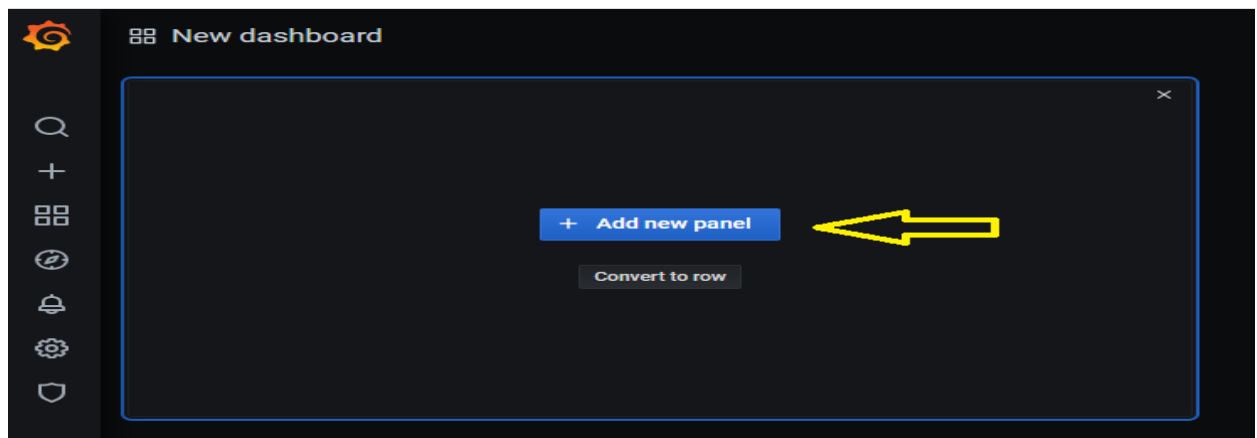
iii- This shows a panel with 2 sections. The top section as seen in the image below contains the chart display area and options for controlling the chart displayed.

- o The yellow arrow shows a time duration of metrics to plot on the chart.
- o The orange arrow shows the time interval for refreshing the chart
- o The light green arrow points to a button that display more options.
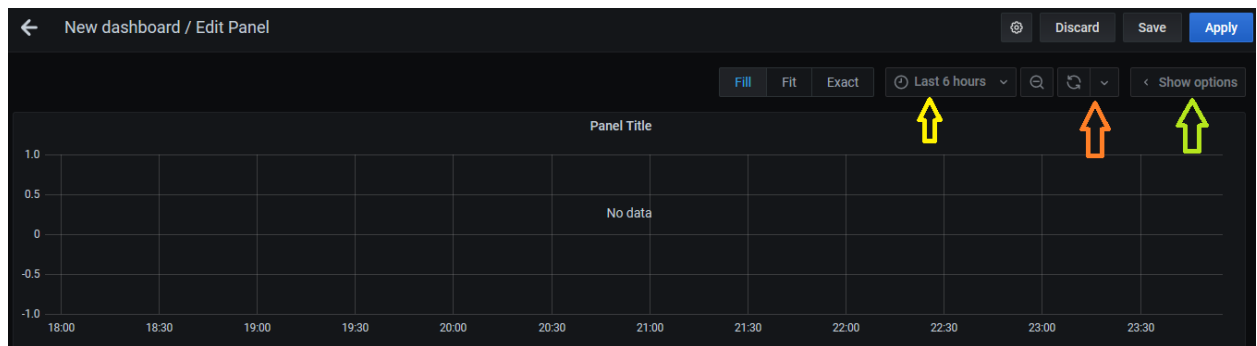
iv- The image below shows the lower section of the panel. This section allows for selecting the data source and metrics with which to plot the chart.
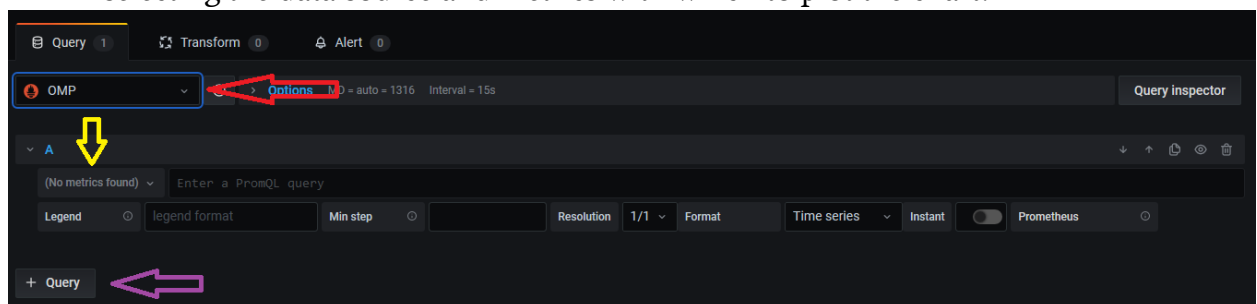
- o The red arrow point to the name of the data source to use. In this case, the Prometheus data source name is OMP.

o The yellow arrow point to the metric selection panel. No metric have been selected.
o The purple arrow points to the query button which lets users add more metrics for plotting the chart.

Example of Grafana Panel for CPU Total Utilization

- **Zebra's Grafana Metrics**

    Zebra provides 4 types of metrics based on a unique prefix system:

1. **TOU_ Prefix**

    Metrics with **TOU_** Prefix contains values of **Total utilization** of an lpar. E.g. **TOU_VIRPT** is a metric containing numeric value of **VIRPT** lpar Total Utilization.

2. **EFU_ Prefix**

    Metrics with **EFU_** Prefix contains values of Effective utilization of an lpar. E.g. **EFU_VIRPT** is a metric containing numeric value of **VIRPT** lpar Effective Utilization.

3. **MSU_ Prefix**

    Metric with **MSU_** Prefix contains System's MSU value E.g. **MSU_VIRPT**.

4. **VC_ Prefix**

    Metrics with **VC_** Prefix contains SYSCPUVC (Percentage of Maximum general purpose processor capacity spent on behalf of a group/class) value. E.g. **VC_TSO** is a metric containing numeric value of **TSO** Group/Class.

App Queries

1. **Monitor 3**

   RMF Monitor III produce realtime reports. Zebra can parse RMF III reports using the /rmfm3 endpoint and also provide filter for CPC, PROC and USAGE reports.

- **Recognised Zebra Query parameters for RMF III**
  i. **report (needed)**

     This parameter takes the title of the report to retrieve from DDS and send a request to DDS through /gpm/filename Endpoint. E.g CPC, PROC, USAGE

  ii. **reports (needed)**

      Not all Monitor III reports are accessible through /gpm/filename Endpoint of DDS. Reports parameter also takes the title of monitor III reports and send a request to DDS through /gpm/reports/filename Endpoint. E.G SYSINFO

  iii. **parm (optional/filter)**

       This url parameter takes a caption parameter(for CPC report) or table parameter (For reports without caption) as value. E.g ALL (returns all caption parameters and their value), CPCHCMSU, PRCPSVCL etc

  iv. **lpar_parms (optional/filter)**

      This is a parameter fof CPC reports and it takes the name of an Lpar (CPCPPNAM value) and returns information about it. If user specifies ALL_CP as its value, it returns information for all lpars.

  v. **Job (optional/filter)**

     This is a parameter of PROC and USAGE reports. It takes the name of a JOB (PRCPJOB or JUSPJOB value) and returns information about it. If user specifies ALL_JOBS, it returns information about all the JOBS.

- **CPC Report**
  a. http://localhost:3090/rmfm3/?report=CPC
  b. http://localhost:3090/rmfm3/?report=CPC&parm=CPCHCMSU
  c. http://localhost:3090/rmfm3/?report=CPC&parm=ALL
  d. http://localhost:3090/rmfm3/?report=CPC&lpar_parms=ALL_CP
  e. http://localhost:3090/rmfm3/?report=CPC&lpar_parms=VIRPT
  f. http://localhost:3090/rmfm3/?report=CPC&lpar_parms=VIRPT&parm=CPCHLMSU

- **SYSINFO Report**
  g. http://localhost:3090/rmfm3?reports=SYSINFO

- **USAGE and PROC Reports**
  h. http://localhost:3090/rmfm3?report=PROC
  i. http://localhost:3090/rmfm3?report=PROC&job=SDSFAUX
  j. http://localhost:3090/rmfm3?report=PROC&job=ALL_JOBS

      k. http://localhost:3090/rmfm3?report=PROC&parm=PRCPSVCL

2. **Post Processor (PP)**

      Zebra also parse RMF I (post processor report) and produce JSON for workload and CPU reports. It also provide filter parameters for Workload Post processor report.

- **Recognized PP parameters**

vi.   **Report (needed)**

    This takes the title of report to retrieve from DDS. EG CPU, WLMGL

vii.   **Date (needed)**

    This takes the date of the report to retrieve from DDS. It takes both the start date and End date separated by a comma. The date is in the format (YYYYMMDD). E.g date=20200819,20200820

i.   **SvcCls (optional/filter)**

    This parameter takes the name of the service class for which to retrieve information about. E.g STCHIGH

ii.   **Wlkd (optional/filter)**

    This parameter takes the workload class name for which to retirieve information about. E.g TSO

iii.   **Duration (optional/filter)**

    This parameter takes the duration for which to retrieve information. It takes both the start and end time separated by a comma. The time is in the format HH.MM.SS

iv.   **Time (optional/filter)**

    This parameter takes the exact time for which to retrieve information. The time is in the format HH.MM.SS

- **Workload**

    a. http://localhost:3090/rmfpp?report=WLMGL&date=20200731,20200731

    b. http://localhost:3090/rmfpp?report=WLMGL&date=20200731,20200731&SvcCls=STCHIGH

    c. http://localhost:3090/rmfpp?report=WLMGL&date=20200731,20200731&SvcCls=STCHIGH&Time=05.30.00

    d. http://localhost:3090/rmfpp?report=WLMGL&date=20200731,20200731&SvcCls=STCHIGH&duration=05.30.00,09.30.00

    e. http://localhost:3090/rmfpp?report=WLMGL&date=20200731,20200731&Wlkd=TSO

    f. http://localhost:3090/rmfpp?report=WLMGL&date=20200731,20200731&Wlkd=TSO&Time=05.30.00

    g. http://localhost:3090/rmfpp?report=WLMGL&date=20200731,20200731&Wlkd=TSO&duration=04.00.00,07.30.00

- **CPU**

    h. http://localhost:3090/rmfpp?report=CPU&date=20200731,20200731

3. Static

    Zebra can also parse static Post processor XML file into JSON. The user needs to use /static endpoint and provide the path of the xmlfile to as a value of file url parameter. The user also need to specify the title of the report as a value for type parameter. Eg CPU(cpu report) or WLM(workload report)

    a. http://localhost:3090/static?file=C:\Users\Salis\Desktop\rmfpp.xml&type=CPU

Contributor Documentation

Project Description

Zowe is a great systems operations tool. One of the systems programmers or performance analyzer's job is to decode SMF/RMF reports to check system's health. Having a generic parser for SMF datasets and/or RMF (or CMF) reports for Zowe would open various opportunities to create/re-use many open-source monitoring tools out there.

Use Case

The Project have five (5) Use Cases:
1. Parsing RMF Monitor III Report to JSON
2. Parsing RMF Monitor I Report to JSON
3. Parsing RMF static XML File to JSON
4. Saving RMF Monitor III Report to MongoDB
5. Plotting RMF Realtime Metrics with Grafana
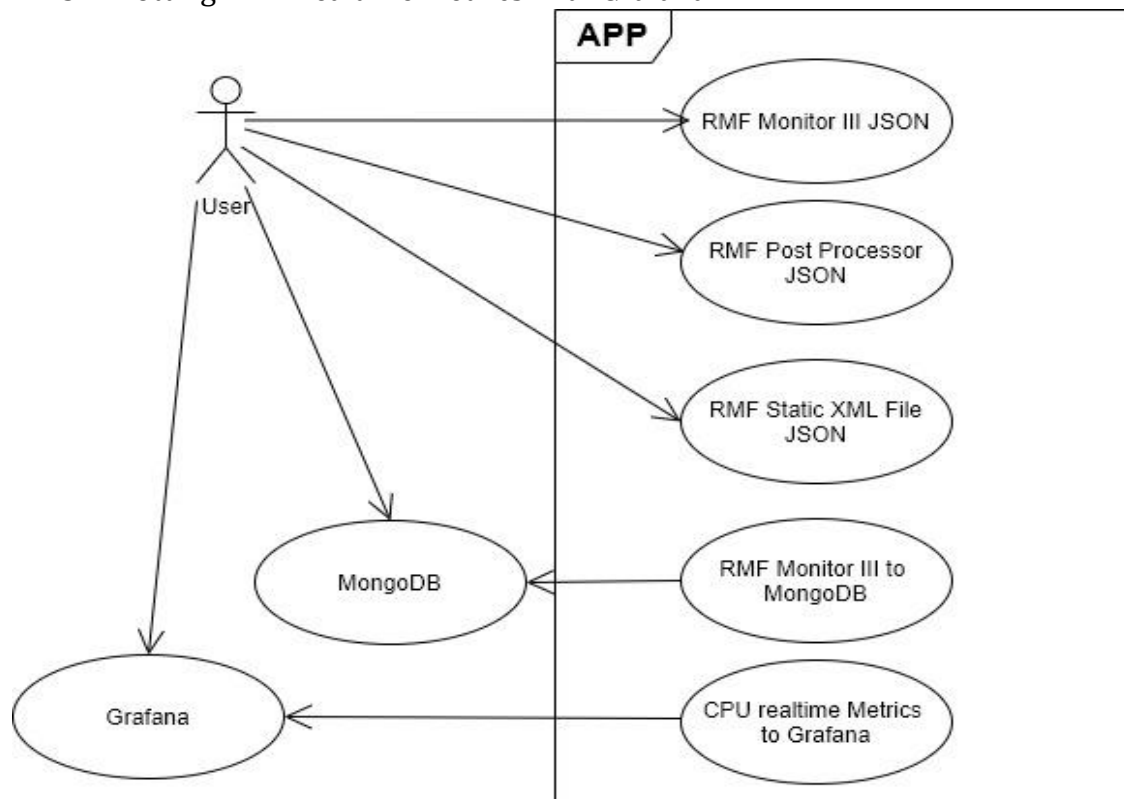
Figure 1: Use Case Diagram

User Requirement

- The Project should create a Parsing Engine for SMF/RMF Report.
- Parsing Engine Output should be in JSON/CSV Format.
- Parsing Engine Output should be saved to NoSQL/Streaming database.
- The Project should provide Graphical System performance Monitoring.

System Environment
- **Development:** Visual Studio Code.
- **Runtime Environment:** Node version 8.11.2 | NPM 5.6.0.
- **Database:** MongoDB.
- **Database Management:** MongoDB Compass.
- **Server:** Ubuntu 18 LTS.
- **Graphical tool:** Grafana.
- **Grafana Data Source:** Prometheus.
- **Unit Testing:** -.
- **Diagrams:** Draw.io

Design Approach

The design approach used in the project is based on the following:
- **Data Flow Design**

    SMF/RMF data is retrieved over the internet in XML format. The XML is then passed to the parsing engine and an output in JSON/CSV format is produced. The outputs for Realtime RMF reports are saved into a NoSQL database. Performance metrics from Realtime reports are exposed through an endpoint and Scraped Using Prometheus.

- **Architecture Design**

    The project will follow a Three Layer Architecture so that the objects in the system as a whole can be organized to best separate concerns and prepare for distribution and reuse.

- **Graphical Tool**

    Prometheus serves as a data source for building Realtime dashboards using a graphical tool. The project makes use of an open source graphical tool (Grafana) for creating real time monitoring dashboards from parsed RMF data.

Design Pattern:

The Application Classes were factored into the following 3 layers:
i.   **The App-Server Layer**

    This layer consist of the following components:
    a. **HTTP GET Functions**

        These functions send a HTTP GET Request to RMF DDS server for RMF Monitor III or RMF Post Processor data.
    b. **Parser**

        This consist of functions for parsing RMF Monitor III and RMF Post Processor data.
    c. **Model**

This consist of Schema definitions for data to be saved into MongoDB.

ii. **The data Layer**
This layer contains the two data warehouse for the project:
a. **Prometheus**
Prometheus saves Realtime Performance metrics exposed by the parsing Engine.
b. **MongoDB**
MongoDB saves Realtime data output from the parsing Engine.

iii. **The Presentation Layer**
This layer consist of:
a. **Zowe API Catalog/Browser**
This tool displays JSON output from the parsing Engine.
b. **Grafana**
This tool displays dashboard from real time data output from the parsing Engine.

System Design Consideration
1. **Directories**
   i. **Root Directory**
   This directory consist of:
   - **App.js file**
   - **Zconfig.json**
     This file contains the configurations of the parsing engine in JSON format.
   - **Cpurealtime.js**
     This file contains function that expose real time CPU utilization metrics using prom-client library.
   - **Mongo.js**
     This file contains functions that save real time parsing engine output to MongoDB.

   ii. **App Server Directory**
   This directory Consist of:
   - **Controllers folder**
     This folder is made up of files that contain functions controlling the Events/Actions of the parsing Engine.
   - **Parser folder**
     This folder is made up of files containing functions for parsing real time and post processor data by the parsing Engine.

- **Models folder**

   This folder contains Schema files for saving data to MongoDB as well as db.js file which contains functions for connecting to MongoDB.

- **Routes folder**

   This folder contain files for mapping URL Endpoints to controller functions of the parsing Engine.

2. **Exception Handling**

   Exception Handlers occur at the application level. Errors are displayed in JSON format.

<u>Architecture</u>

1) **User (request):** User send a request to the zebra App using any of its recognized URL(s).
2) **GetRequest**: zebra app send a get request to RMF DDS server for post processor or Monitor III data depending on the URL specified by the user. This happens through the use of DDS HTTP API. DDS server returns an XML file.
3) **Parser:** The XML file returned is feed to RMF post processor parser, RMF monitor III parser or CPU utilization parser depending on the URL specified by the user. The parser returned a JSON.
4) **User (response):** User can view parsed RMF report using a browser or Zowe API Catalog.
5) **Prom-client:** The JSON returned by CPU utilization parser is used to create custom Prometheus metrics using prom-client library. The custom Prometheus metrics are exposed via /prommetric endpoint by the zebra app.
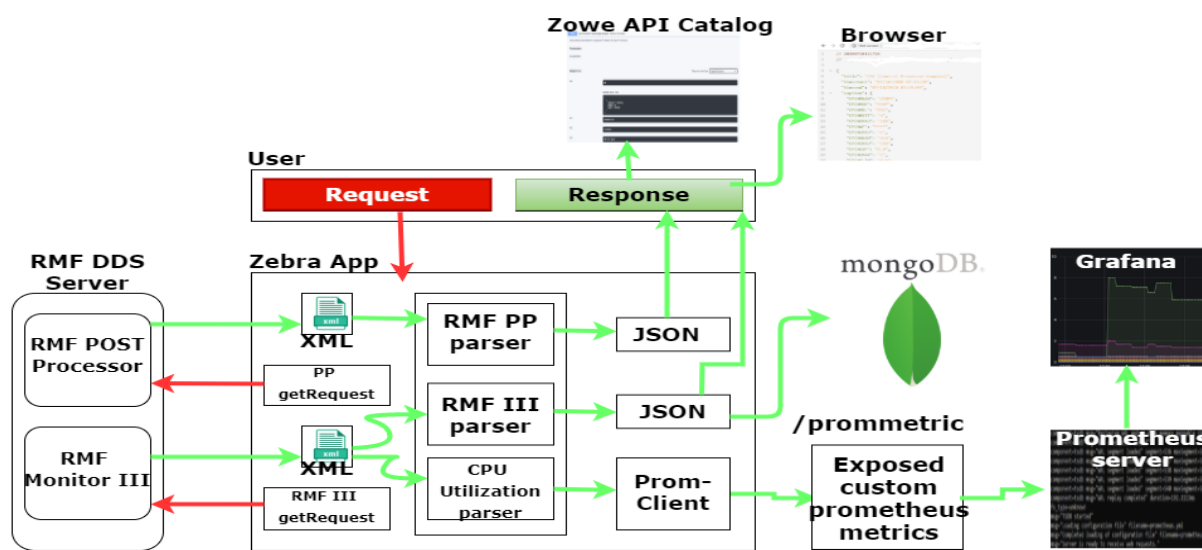
Figure 2: Project Architecture

6) **Prometheus server:** this server scrape custom Prometheus metrics from zebra app /prommetric endpoint
7) **Grafana:** Grafana dashboards are built by connecting Grafana to Prometheus server. This dashboard shows CPU utilization chart I Realtime.
8) **MongoDB**: Realtime data output from the parsing engine is saved to MongoDB database.

Class Diagram
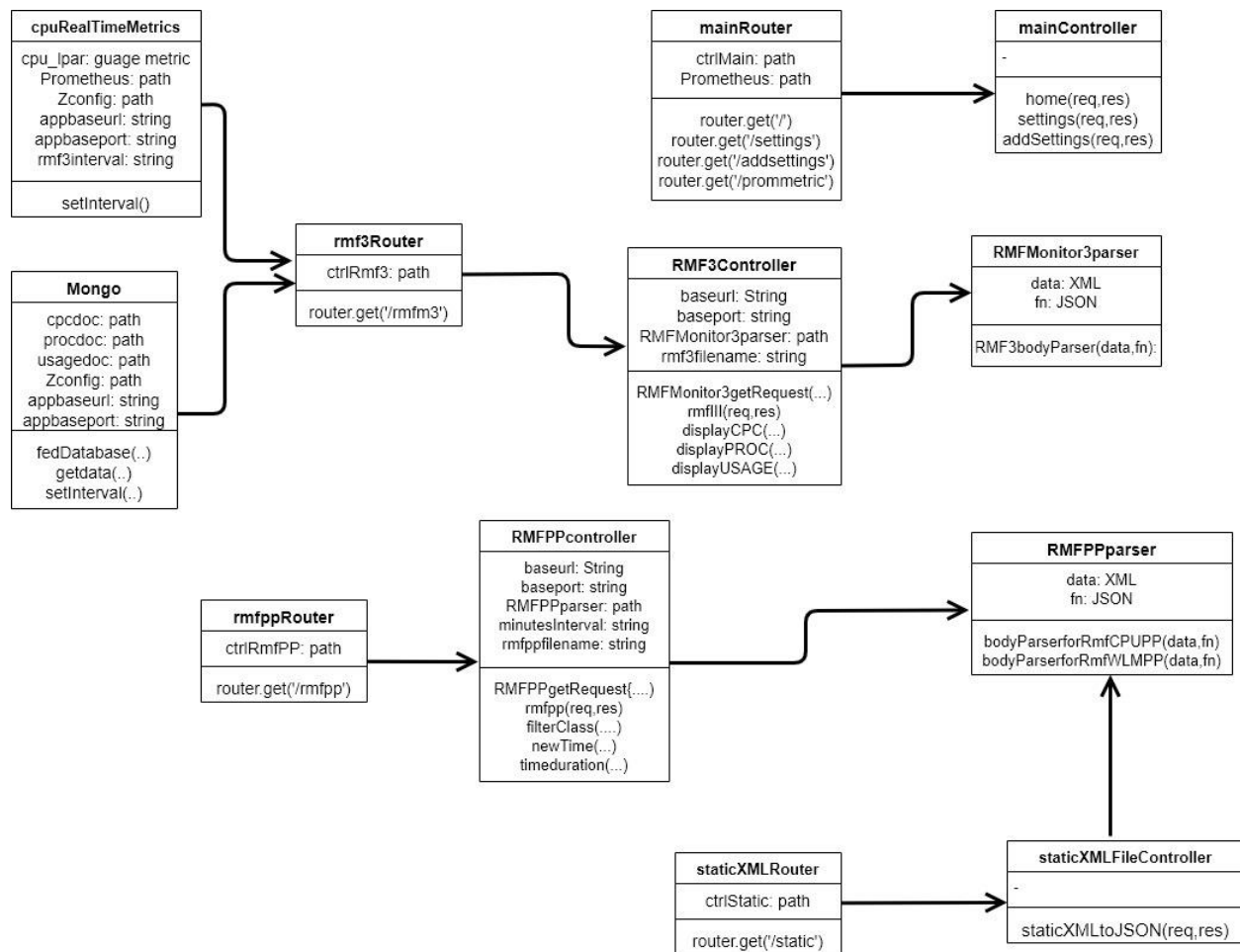
Figure 3: Project Class Diagram

The class diagram shows directional association between the project classes. Classes are represented in the project by files with ".js" as file extension.

Routers classes are the starting point of communication with other classes as they map incoming API request from users to controller and parser classes. The Router class names

contains the word "Router" at the end. These classes are responsible for recognizing the app's endpoint.

Controller class names contains the word "Controller" at the end and consist of functions for:
- Sending GET Request to DDS server
- Sending XML to Parser
- Filtering parsed JSON based on user specified parameters in the URL
- Displaying the JSON response to User
- Adding Configuration settings to the App

Parser classes consist of functions for parsing XML to JSON. RMF monitor III parser has a single function which can parse all RMF III XML reports due to their format consistency. Monitor I parser (RMFPPparser) has two functions for parsing CPU post processor and Workload Postprocessor data.

Mongo and **cpuRealTimeMetrics** classes have a **setInterval()** function which makes them run continuously at an interval specified by the user in the app configuration. They both interact with monitor III router to retrieve a JSON.

Mongo class is responsible for retrieving real time **CPC**, **PROC** and **USAGE** monitor III reports and fed them into **MongoDB** while cpuRealTimeMetrics class is responsible for retrieving real time **CPC** report **JSON**, filtering the JSON for **Total utilization**, **MSU value** and **Effective Utilization values**. It then use prom client library to create custom gauge metric. The custom metrics are saved into a register. The metrics in the register are exposed via an Endpoint in the **mainRouter** class. The exposed metrics can then be scraped by Prometheus and real time charts can be plotted using Grafana.

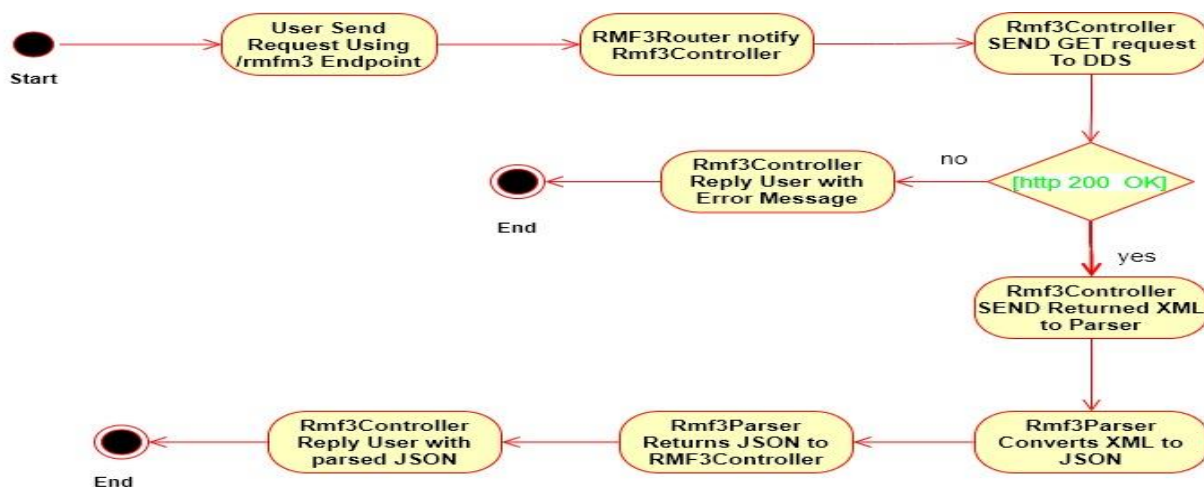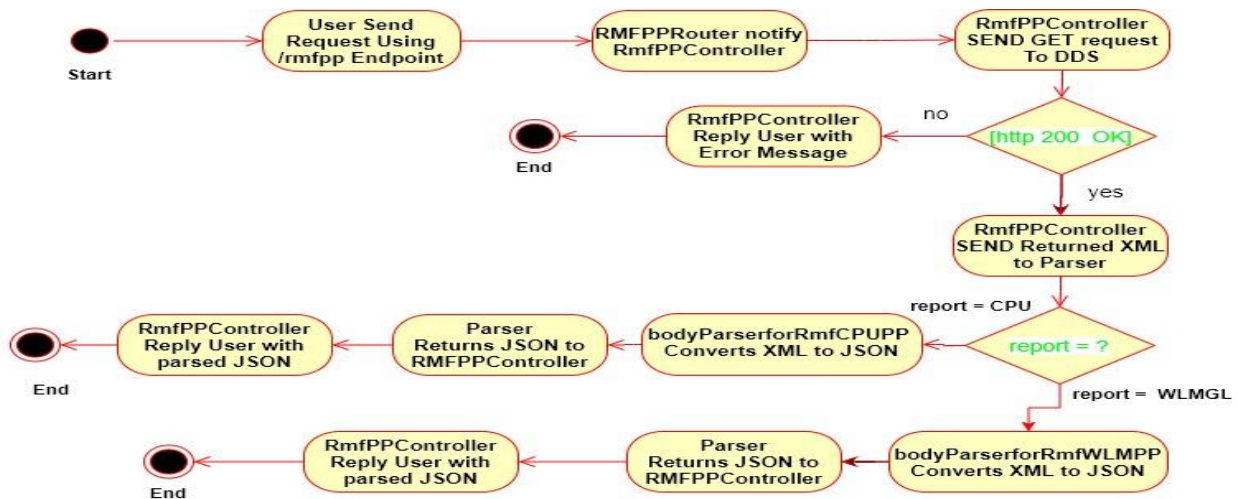## Activity Diagram
i.   RMF Monitor III Activity Diagram

Figure 4: RMF 3 Activity Diagram

RMF Monitor III Activities starts when a User trigger a request using the /rmfm3 Endpoint, This leads to a series of activities involving RMF3Router, RMF3Controller and RMF3Parser. A Condition exist to check is the Request to DDS server is successful, this condition determines the data that get returned to RMF3Controller and finally to the User. In the End, the user receives a JSON Response containing parsed RMF III report or Error Message in case of a failed request to DDS.

ii. RMF Post Processor Activity Diagram

Figure 5: RMF Post Processors Activity Diagram

RMF Monitor I Activities starts when a User trigger a request using the /rmfpp Endpoint, This leads to a series of activities involving RMFPPRouter, RMFPPController and RMFPPParser. Two conditions exist in the activity flow of RMF post Processor. First is the condition that checks if the request to DDS server is successful. The Second one checks the value of the Report parameter specified by the user during a request. The value of the report parameter is used to determine the parser for the returned by the request to DDS.

iii. Static XML File Activity Diagram

Static XML File to JSON Activities starts when a User trigger a request using the /static Endpoint, This leads to a series of activities involving staticXMLRouter, staticXMLFileController and RMFPPParser. A re-use of the RMFPPParser occurs here. Two conditions exist in the activity flow of static File to JSON as well. First is the condition that checks if reading the static file specified by the user in the URL's file (takes file path as value) parameter is successful. The Second one checks the value of the Report parameter specified by the user during a request. The value of the report parameter is used to determine the parser for the returned by the request to DDS.
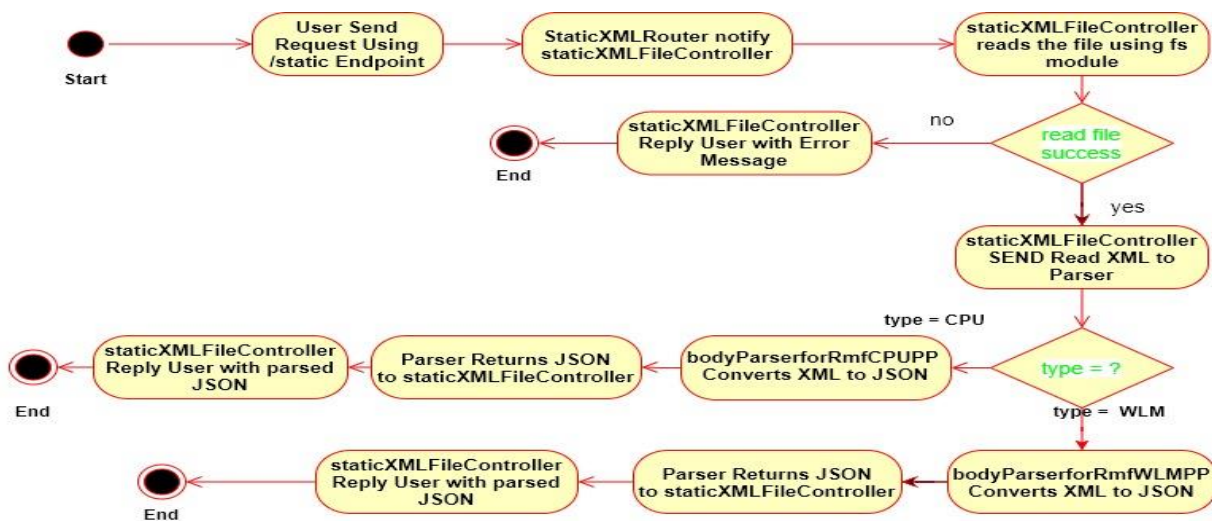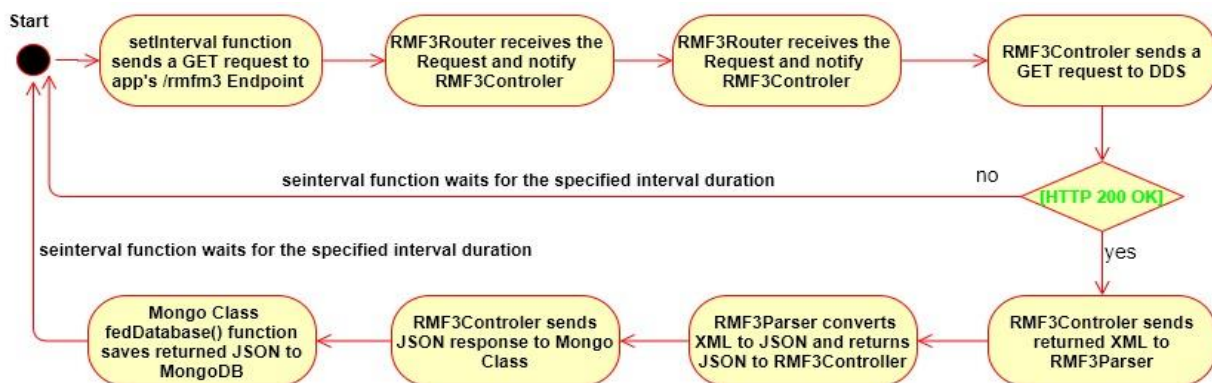
Figure 6: Static XML File Activity Diagram

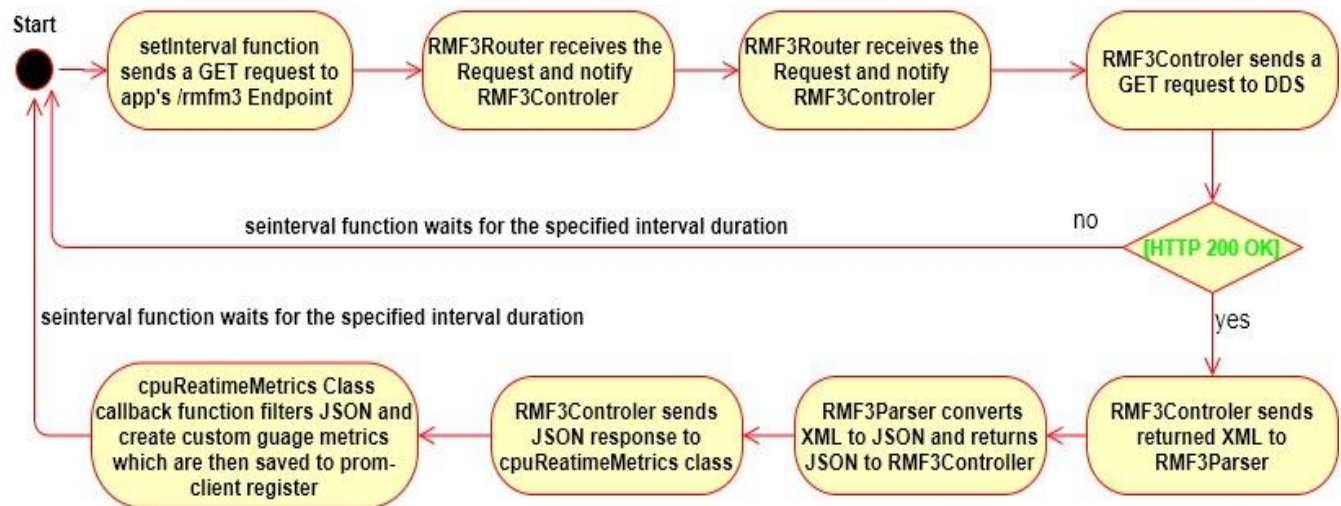### iv.   Mongo Activity Diagram

Figure 7: Mongo Activity Diagram

Activities of the Mongo class do not require user intervention. A setInterval function runs continuously at an interval specified in the Apps Configuration. This function serves as a trigger and make use of RMF III Activities to retrieve and store JSON into MongoDB.

v.   CPU Realtime Metrics Activity Diagram

Just like in Mongo Class, Activities of the CPU Realtime Metrics (cpuRealTimeMetrics) class do not need user intervention. The setInterval function triggers RMF III Activities which returns a JSON. Through a call back function, CPU Realtime Metrics class filters the JSON and create Prometheus custom gauge metrics for Effective utilization, MSU and Total utilization values. These Metrics are then saved to a prom-client library register.

Figure 8: CPU Realtime Metrics Activity Diagram

<u>**Sequence Diagram**</u>

i.   RMF Monitor III Sequence Diagram

RMF monitor III Service is triggered when user send a request using /rmfm3 Endpoint (e.g. localhost:3000/rmfm3?report=CPC).

RMF3Router receives the request and notify RMF3Controller. rmfIII() Function of the controller sends A HTTP API GET Request to DDS Server. This Returns an XML which is then sent to RMFMonitor3Parser. The parser then parse the XML into JSON and returns the JSON to the controllers rmfIII function. This JSON is finally sent to user through Express Response.
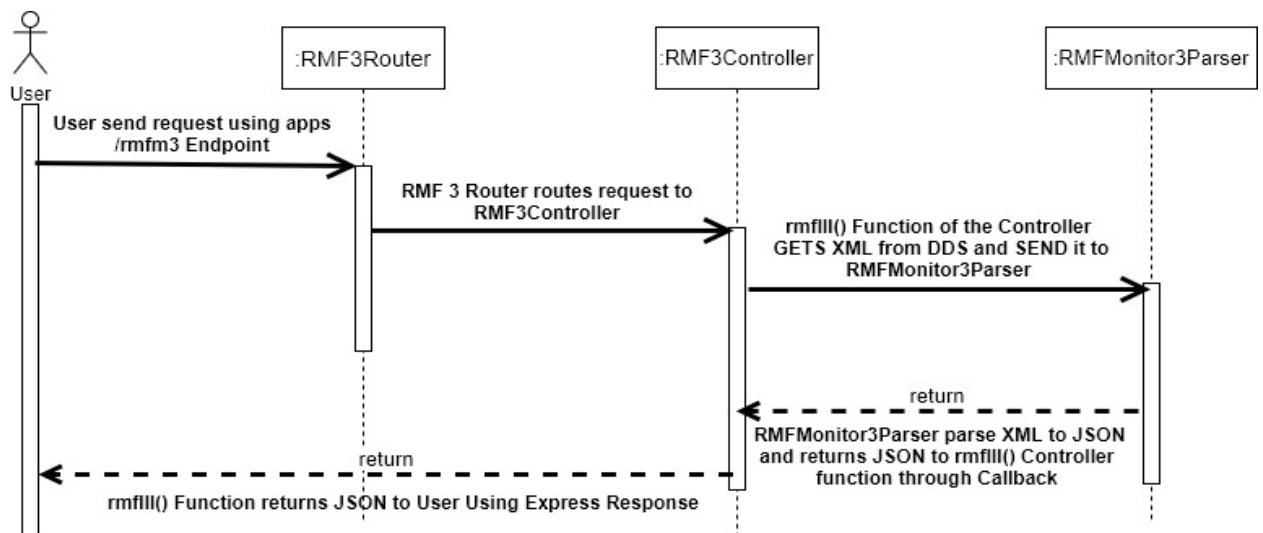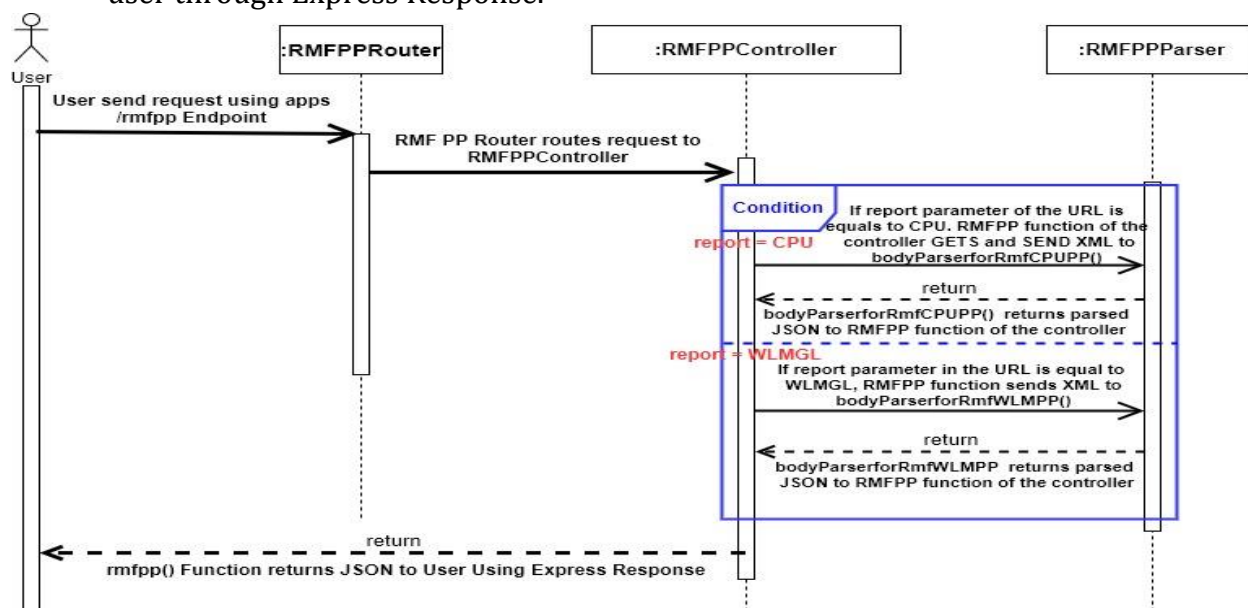

Figure 9: RMF 3 Sequence Diagram

ii.   RMF Post Processor Sequence Diagram

RMF pp service is also triggered when a user send a request using /rmfpp Endpoint (e.g. localhost:3000/rmfpp?report=CPU).

RMFPPRouter receives the request and notify the controller. Rmfpp() function of the controller evaluates the value of report parameter(CPU or WLMGL). It then sends a HTTP GET request to DDS server. The returned XML is send to the appropriate parser(bodyParserforRmfWLMPP or bodyParserforRmfCPUPP) based on the value of the report URL parameter. The parser then parse the XML into JSON and returns the JSON to the controllers rmfpp function. This JSON is finally sent to user through Express Response.

Figure 10: RMF PP Sequence Diagram

iii.   Static XML File

The static XML file service is triggered when a user sends a request using the apps /static Endpoint (e.g. localhost:3000/static?file=/home/salis).

staticXMLRouter receives the request and notify the controller. staticXMLtoJSON () function of the controller evaluates the value of type parameter(CPU or WLM). The controller re-use rmfppparser for parsing XML to JSON. The XML file specified in the URL file (takes file path) parameter is send to the appropriate parser (bodyParserforRmfWLMPP or bodyParserforRmfCPUPP) based on the value of the type URL parameter. The parser then parse the XML into JSON and returns the JSON to the controllers staticXMLtoJSON function. This JSON is finally sent to user through Express Response.
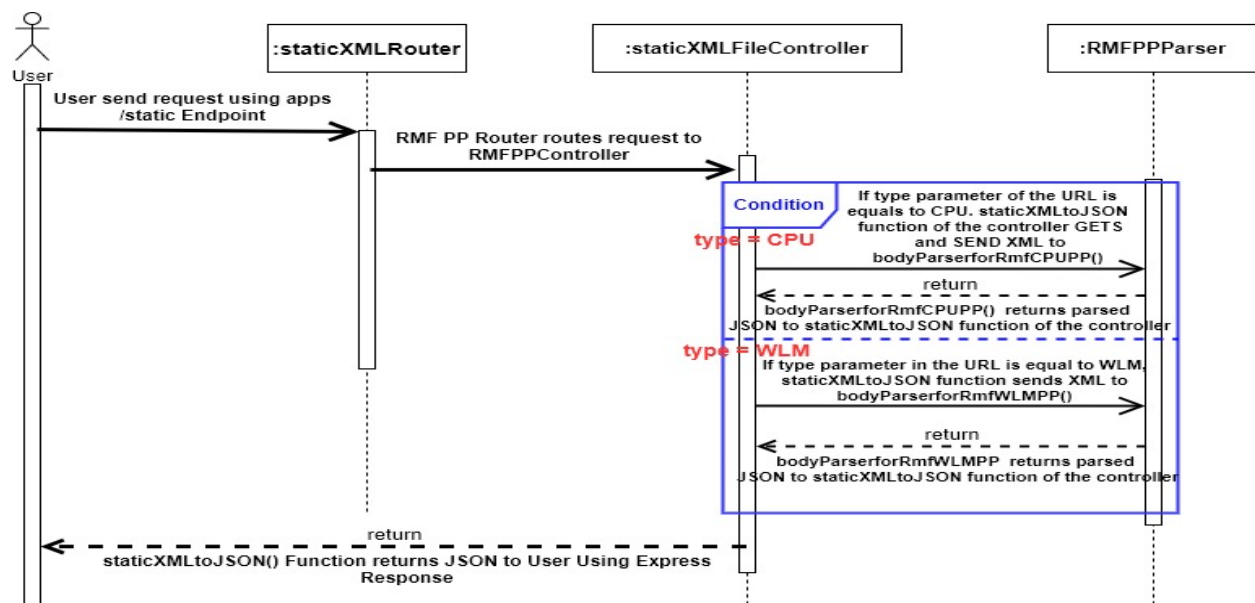
Figure 11: Static XML File Sequence Diagram
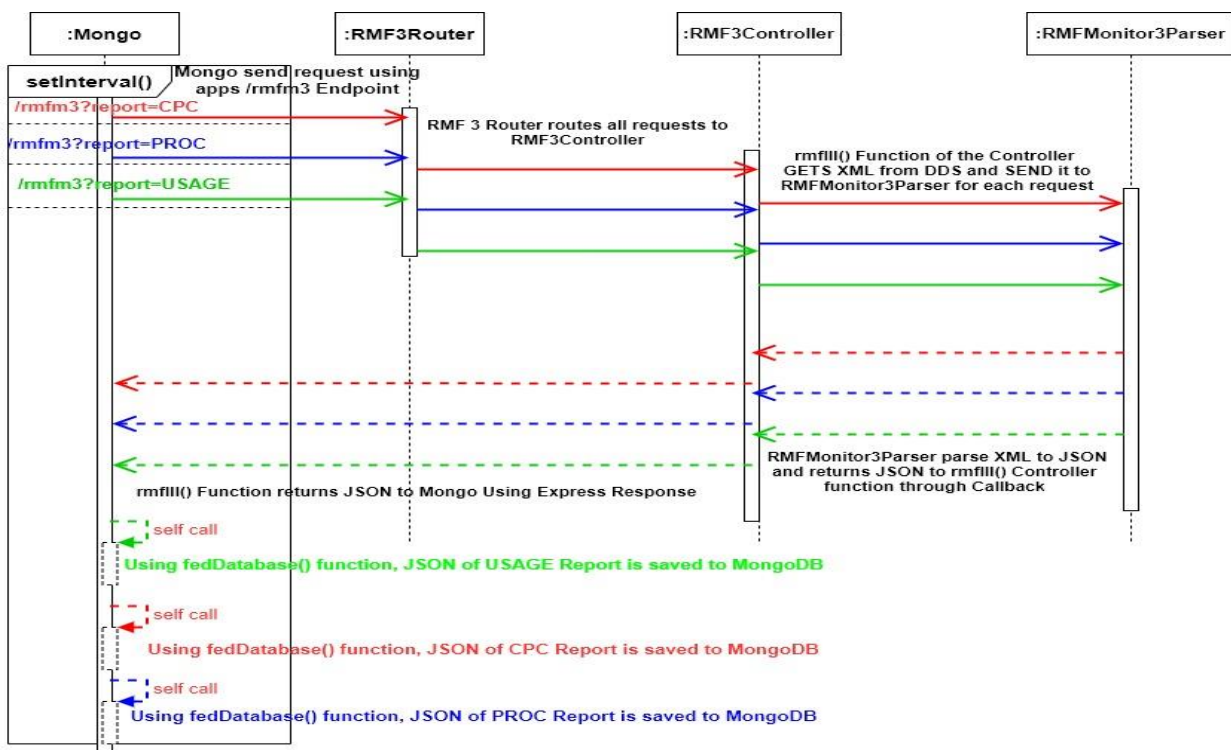
iv.   **Mongo**

Figure 12: Mongo Sequence Diagram

Unlike the first three services, Mongo service does not require a trigger from user. The class contains a setInterval function that runs continuously based on the interval specified in the App configuration.

When setInterval function is activated, it send three request for CPC, PROC and USAGE JSON report by triggering the apps /rmfm3 endpoint. For each report type (CPC, PROC And USAGE), RMF3Router sends a notification to RMF3Controller. rmfIII function of the controller sends a HTTP GET request to DDS for each report and the returned XML is sent to RMFMonitor3Parser. The parser parse each report and return 3 JSON to the rmfIII controller function. All 3 JSON's are sent to Mongo class.

Using the fedDatabase() function of the Mongo class, The 3 JSON response are saved to the appropriate documents in MongoDB.

v.   CPU Realtime Metrics

The CPU real-time class also does not need user intervention. It's responsible for creating and saving Prometheus Custom metrics that can be used to plot real-time graphs using Grafana.
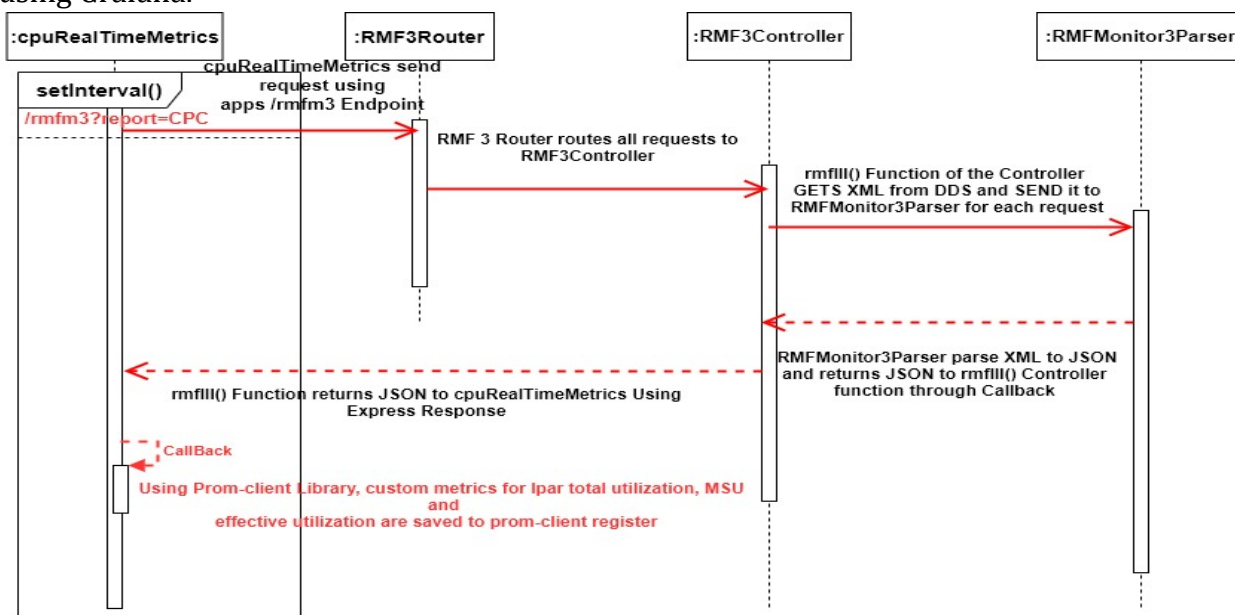
Figure 13: CPU Realtime Metrics Sequence Diagram

The setInterval function of the CPU real-time metrics class sends a request to the app /rmfm3 for CPC JSON. The process of retrieving the JSON is similar to Mongo class. Data flow from router, controller and parser of the RMF Monitor III service. The JSON returned to CPU real-time metrics class is processed using a callback function. The JSON is filtered for Effective utilization, MSU and Total Utilization Values. These values are used to create custom gauge metrics using prom-client library. These metrics are saved into a register.

Entity Diagram

The Project make use of MongoDB. Mongoose library was used to structure the data. In MongoDB each entry in a database is called a document. In MongoDB a collection of documents is called a collection (think "table" if you're used to relational databases). In Mongoose the definition of a document is called a schema.
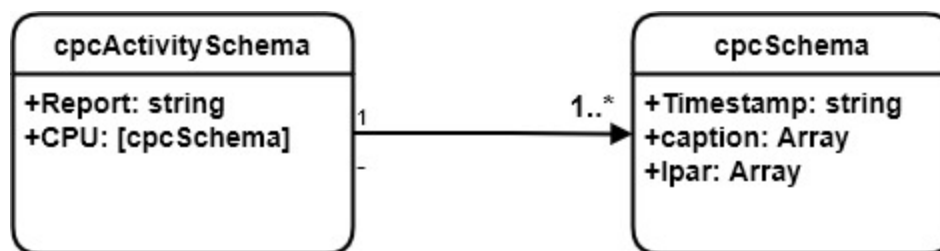
1. **CPC Activity**

Figure 14: CPC Schema Definition

For CPC report, the main document definition is named cpcActivitySchema. The main document contains two fields:

- **Report:**

    Report is a string that represent the Document title

- **CPU**

    CPU field represent (is populated by) a subdocument. The subdocument is named cpcSchema and have 3 fields:

    - **Timestamp**

        This is a string that represent the CPC report **timestart**.

    - **Caption**

        This is an array and is populated by CPC report **caption** key.

    - **Lpar**

        This is an array and is populated by an array of the CPC report **table** key

The Mongo Class Create CPC Activity document only once and continue to populate CPU field of the main document by the subdocument.
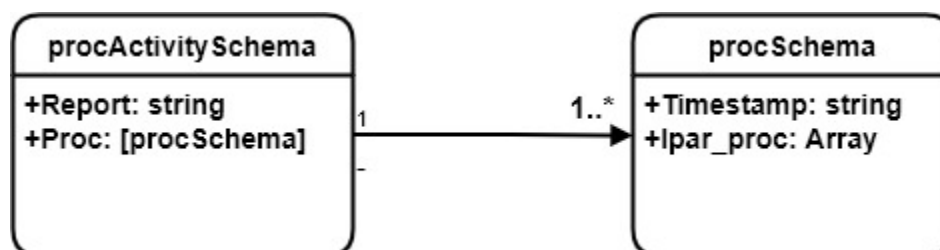
2. **PROC Activity**

Figure 15: PROC Schema Definition

For PROC report, the main document definition is named procActivitySchema. The main document contains two fields:

- **Report:**

    Report is a string that represent the Document title

- **Proc**

    Proc field represent (is populated by) a subdocument. The subdocument is named procSchema and have 2 fields:

    - **Timestamp**

        This is a string that represent the PROC report **timestart**.

    - **Lpar_proc**

        This is an array and is populated by an array of PROC report **table** key

The Mongo Class Create PROC Activity document only once and continue to populate Proc field of the main document by the subdocument.

3. USAGE Activity

For USAGE report, the main document definition is named usageActivitySchema. The main document contains two fields:

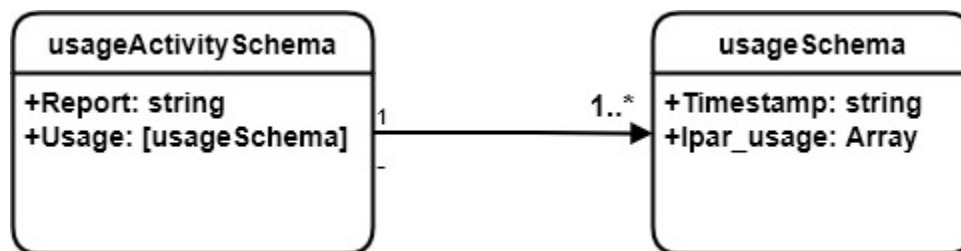| usageActivitySchema		usageSchema
+Report: string +Usage: [usageSchema]	1 ——— 1..*	+Timestamp: string +lpar_usage: Array

Figure 16: Usage Schema Definition

- **Report:**

    Report is a string that represent the Document title

- **Usage**

    Usage field represent (is populated by) a subdocument. The subdocument is named usageSchema and have 2 fields:

    - **Timestamp**

        This is a string that represent the USAGE report **timestart**.

    - **Lpar_usage**

        This is an array and is populated by an array of USAGE report **table** key

The Mongo Class Create USAGE Activity document only once and continue to populate Usage field of the main document by the subdocument.