



ZEBRA – A ZOWE INCUBATION PROJECT

Project Team
Yongkook(Alex) Kim
Salisu Ali
Justin Santer
Andrew Twydell

For:
ZEBRA V1.0.0

Contents:

User Documentation

• How to Run App	1
• App User Interface	3
• How to Configure App	5
• How to Set up TLS	8
• How to Connect ZEBRA to Zowe API ML	9
• How to Connect MongoDB	10
• How to Connect Prometheus	11
• How to Connect Grafana	12
• App Queries	17

Contributor Documentation

• Project Description	21
• User Requirement	21
• System Environment	22
• Design Approach	22
• Design Patterns	22
• Design Consideration	23
• Architecture Design	24
• Class Diagram	25
• Activity Diagram	26
• Sequence Diagram	29
• Entity Diagram	33

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 Node js comes along with npm. Git is also needed for cloning the app from a github repository. Prometheus, MongoDB and Grafana are optional programs needed to run ZEBRA.

Clone ZEBRA

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

- git clone <https://github.com/zowe/ZEBRA>

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:

- cd into the cloned app src folder (if you are not there already).
- 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:

- cd into the cloned app src folder (if you are not there already).
- run nodemon

OMP Mentorship Program 2020: Project ZEBRA

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

ZEBRA All API's Documentation Manage File About

- Browse RMF data for real-time LPAR
- Browse RMF data real-time Workload
- Browse RMF data for historical CPU Reports (post-processor report)
- Browse RMF data for historical Workload Reports (post-processor report)

Select LPAR Name

RMF Monitor III report

Retrieve RMF III Report In JSON Report Title

RMF Monitor I report

Retrieve RMF I Report In JSON Report Title Start Date End Date

RMF Static XML File

Convert RMF XML file to JSON Report File No file chosen

Browse RMF data from MongoDB Browse RMF real-time data with Grafana

This project was part of Open Mainframe Project Mentorship Program

All API's

OMP Mentorship Program 2020: Project ZEBRA

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

RMF Monitor III Convert RMF Monitor III Reports to JSON	
GET	/v1/{lpar}/rmf3/{report} Get RMF III Report by title only
GET	/v1/{lpar}/rmf3/{report}?parm={value} Get RMF III CPC Report by title and caption parameter
GET	/v1/{lpar}/rmf3/{report}?lpar_parms={value} Get RMF III CPC Report by title and lpar name
GET	/v1/{lpar}/rmf3/{report}?parm={value1}&lpar_parms={value2} Get RMF III CPC Report by title, caption and Lpar name
GET	/v1/{lpar}/rmf3/{report}?job={value} Get RMF III PROC and USAGE Reports by title and job name
GET	/v1/{lpar}/rmf3/{report}?resource={resource} Get RMF III SYSINFO Report by title/Reports not accessible through gpm
GET	/v1/{lpar}/rmf3?id={id}&resource={resource} Get RMF Metrics by ID and resource value
GET	/v1/{lpar}/rmf3?id={id} Get RMF Metrics by ID value


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

 ZEBRA All API's Documentation Manage File Settings About

New File

Convert RMF Static XML file to JSON

Report File

Choose File No file chosen

Report Type

CPU

Parse

Recent Files

rmfpp_cpu.xml

Report Type

CPU

Parse

Delete

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)

OMP Mentorship Program 2020: Project ZEBRA

- **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.

The screenshot shows the ZEBRA application interface. At the top, there's a navigation bar with 'ZEBRA', 'All API's', 'Documentation', 'Manage File', 'Settings', and 'About'. Below this, a section titled 'Retrieve data from MongoDB' contains input fields for 'Report Type' (set to 'CPC'), 'Report Date' (set to 'nil'), 'Report Duration' (set to 'nil'), and 'Report Time' (set to 'nil'). A 'Get Data' button is on the right. Below the inputs, a 'Timestamp' of '09/30/2020 06:36:40' is displayed. The main area shows a list of 'Captions' organized in a grid, with each caption followed by a status (e.g., 'CPCHPNAM : VIRPT', 'CPCHMOD : 3907'). Below the captions, there's a table labeled 'Lpar' with columns for various system parameters and their values. The table has 13 columns: CPCPPNAM, CPCPDMSU, CPCPAMSU, CPCPCAPD, CPCPLPNO, CPCPLEFU, CPCPLTOU, CPCPLPMU, CPCPPEFU, CPCPPTOU, CPCPIND, CPCPLPND, CPCPDEDP, and CPCPW. The first row shows values for '*CP' and the second row for 'QCK2'. At the bottom, a blue banner states 'This project was part of Open Mainframe Project Mentorship Program'.

CPCPPNAM	CPCPDMSU	CPCPAMSU	CPCPCAPD	CPCPLPNO	CPCPLEFU	CPCPLTOU	CPCPLPMU	CPCPPEFU	CPCPPTOU	CPCPIND	CPCPLPND	CPCPDEDP	CPCPW
*CP				5.0			0.4	2.6	3.0	CS	5	0	112
QCK2	0	1	NNN	1.0	0.4	0.4	0.0	0.2	0.2	CP	1	0	25

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

The Zconfig file have the following parameters:

1) dds (Data Distributed Server)

This contains DDS information on multiple LPAR's as JavaScript objects. The JavaScript object contains:

a. Key

The key will be the LPAR name and this key will be used as `{lpar_name}` in the new API structure.

b. Value

The value for each key (LPAR name) a JSON containing:

- o **ddshhttpype (Distributed Data Server http type)**

This is the hypertext transfer protocol type of the DDS. Its value is either http or https.

ddshttptype: **http**

- **ddbseurl: (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**

- **ddbseport: (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**

- **ddsauth (Distributed Data Server Authentication)**

This config parameter determines the type of connection to DDS, 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 DDS. E.g

ddsauth: **false**

- **ddsuser: (Distributed Data Server USER ID)**

This is the username with which ZEBRA will connect to DDS if value of **ddsauth** is set to true. E.g.

ddsuser: **userID**

- **ddspwd: (Distributed Data Server Password)**

This is the password with which ZEBRA will connect to DDS if value of **ddsauth** is set to true. E.g.

ddspwd: **Password**

- **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**

- **PCI: ()**

This PCI parameter represent the PCI value of the Mainframe. E.g.

PCI: **2091**

- **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.

usePrometheus: **true**

2) **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

ppminutesInterval: 30

3) rmf3interval: (RMF Monitor III Report Interval)

This is the Interval for which DDS Produce RMF III report. E.g. Every 100 seconds

rmf3interval: 100

4) use_cert: (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.

use_cert: true

5) ZEBRA_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.

ZEBRA_httptype: http

6) 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

7) 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

8) 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

9) 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

10) dbname: (Database Name)

This is the name of the database for which RMF III data will be saved. E.g.

dbname: ZEBRA

11) mongoport: (Mongo DB Port)

This is the port number of MongoDB to which RMF III data will be saved. E.g.

mongoport: 27017

12) 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.

useDbAuth: true

13) dbUser: (Database Username)

This is mongodb username with which ZEBRA will connect to database if value of useDbAuth is set to true. E.g.

dbUser: salisuali

14) dbPassword: (Database Password)

This is mongodb password with which ZEBRA will connect to database if value of useDbAuth is set to true. E.g.

dbPassword: salisu

15) authSource: (Authentication Source)

This is mongodb database which contains the username and password for authentication. The default is mongodb's "admin" database. E.g.

authSource: admin

16) grafanahttptype: (http type of running Grafana instance)

This is the hypertext transfer protocol type of the running Grafana instance. Its value is either http or https. E.g.

grafanahttptype: http

17) grafanaurl: (Running Grafana IP)

This is the IP address of running grafana instance. E.g. 127.0.0.1.

grafanaurl: 127.0.0.1

18) grafanaport: (Running grafana Port)

This is the Port of a running grafana Instance. E.g. 3000

grafanaport: 3000

HOW To Configure APP

Users can configure ZEBRA by

1. Editing the Zconfig.json File in an Editor/IDE.

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. Use_cert

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 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. ZEBRA_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.

api-layer > config > local > api-defs				
	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

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**

- **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:

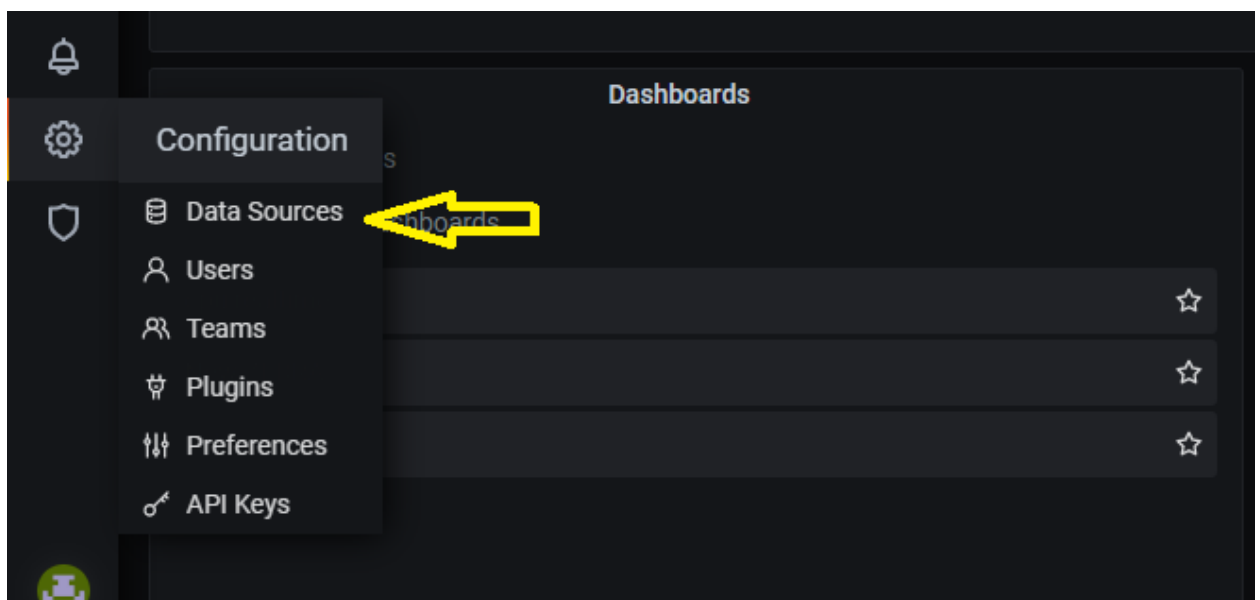
Note: Modify the value for target to point to IP address and port of your running ZEBRA Instance.

How to Connect Grafana to Prometheus

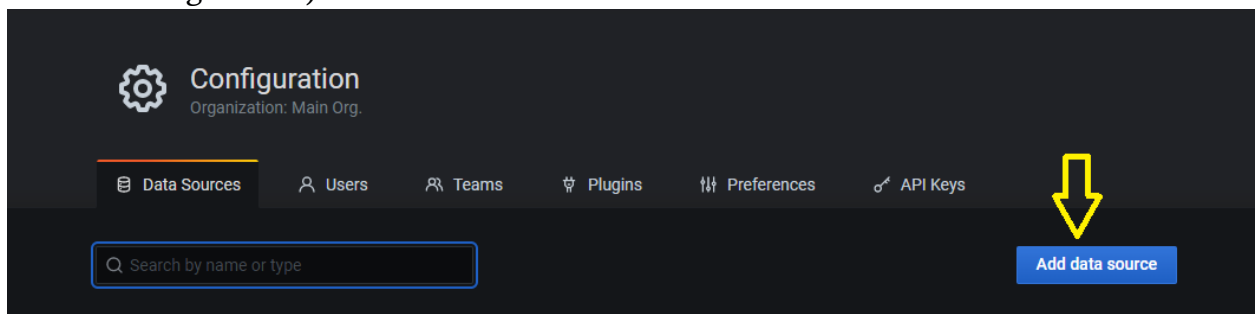
When Prometheus has been connected to ZEBRA, the values scraped by Prometheus can be used to create Realtime dashboards using Grafana.

- 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.

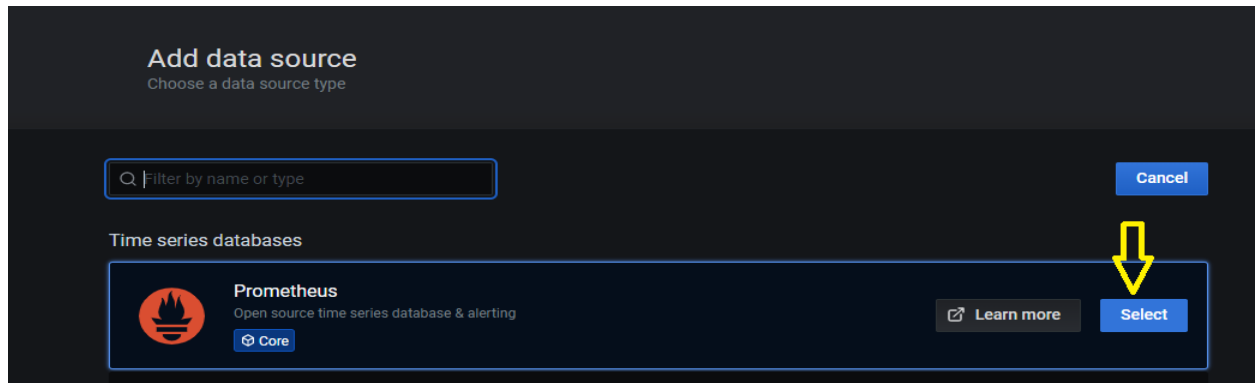


- i. After logging into Grafana, click on the **configuration button** and select **Data source** (as shown by the yellow arrow in the image above).
- ii. Click on **Add data source** button (as shown by the yellow arrow in the image below)

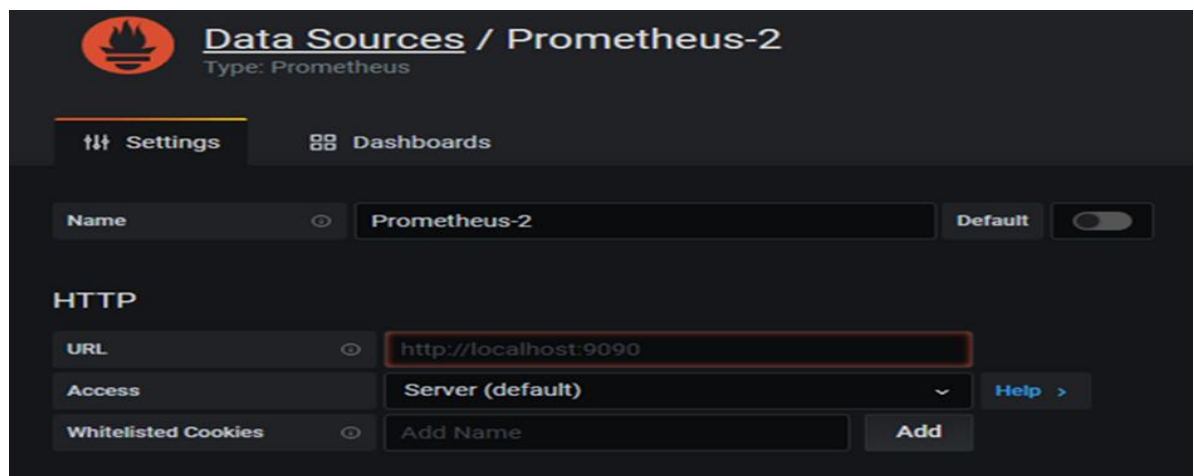


OMP Mentorship Program 2020: Project ZEBRA

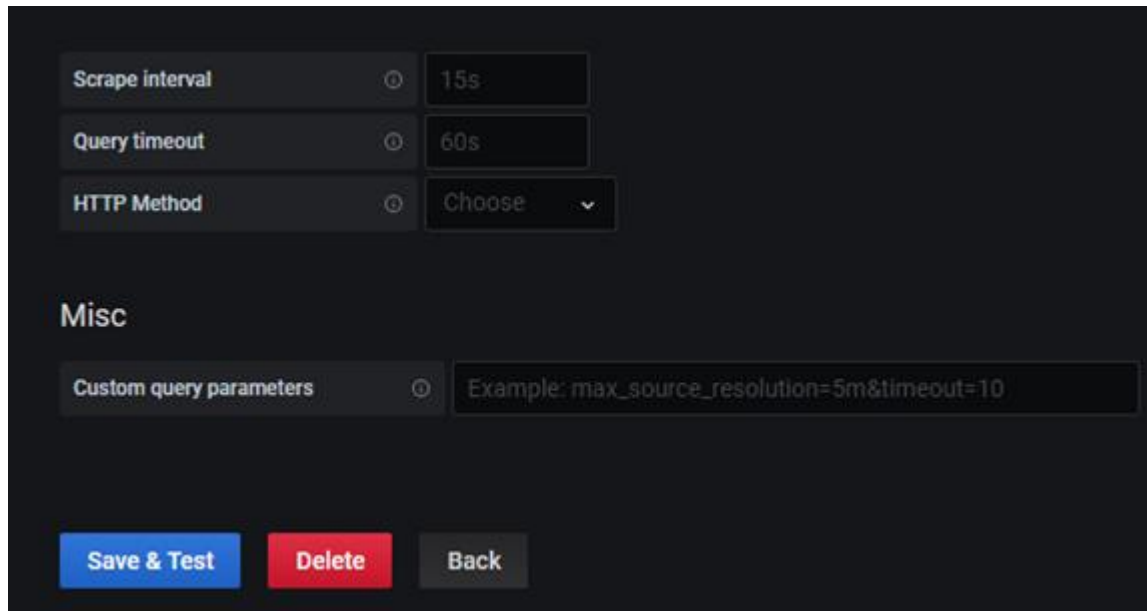
- iii. Hover over Prometheus option and click on **select** (as shown by the yellow arrow in the image below)



- 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.



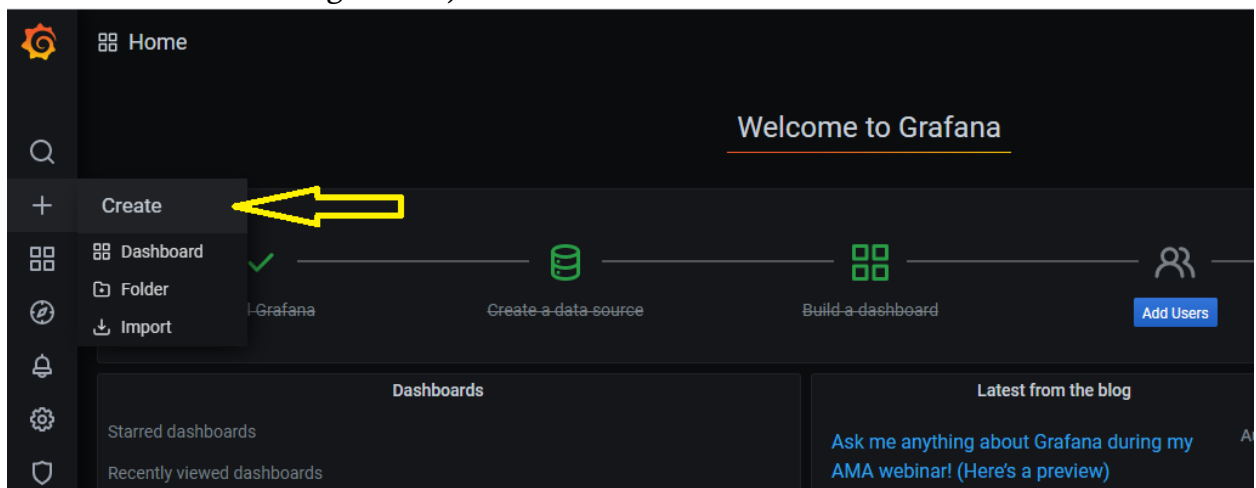
The image shows a configuration interface for a data source in Grafana. It has a dark theme. At the top, there are three settings: 'Scrape interval' set to '15s', 'Query timeout' set to '60s', and 'HTTP Method' set to 'Choose' with a dropdown arrow. Below these is a 'Misc' section with a 'Custom query parameters' field containing the example text 'Example: max_source_resolution=5m&timeout=10'. At the bottom, there are three buttons: 'Save & Test' (blue), 'Delete' (red), and 'Back' (grey).

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

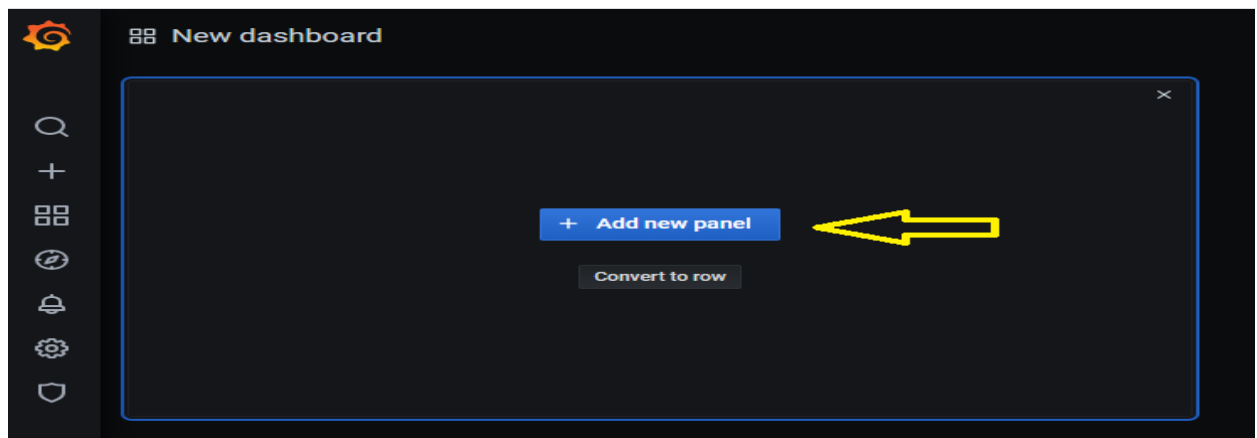
- **Build Your own Dashboard**

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:

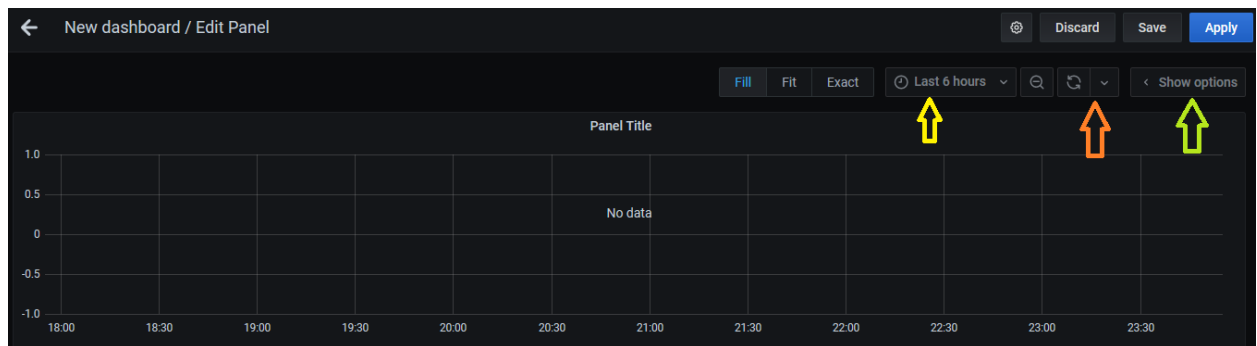
- i- From Grafana homepage, click on create “+” button (as shown by the yellow arrow in the image below)



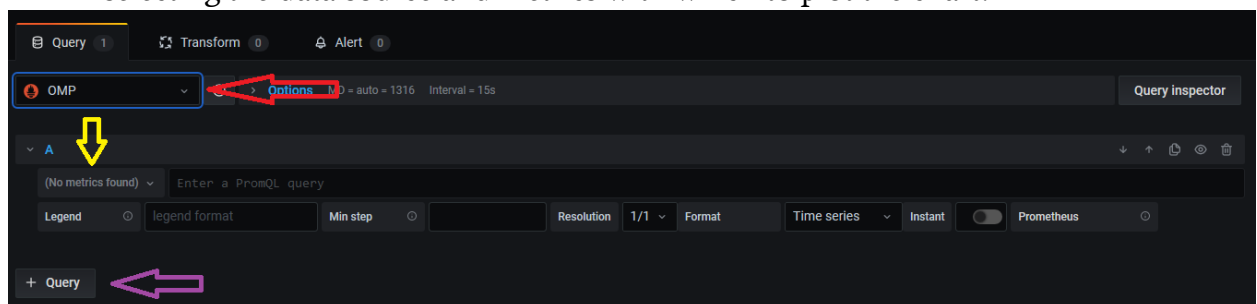
- ii- Click on Add new panel button (as shown by the yellow arrow in the image below)



- 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.
- The yellow arrow shows a time duration of metrics to plot on the chart.
 - The orange arrow shows the time interval for refreshing the chart
 - 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.



- The red arrow point to the name of the data source to use. In this case, the Prometheus data source name is OMP.

OMP Mentorship Program 2020: Project ZEBRA

- The yellow arrow point to the metric selection panel. No metric have been selected.
- 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

Users can now collect metrics for one or more lpar(s) of their choice and plot Grafana charts for the exposed metrics.

ZEBRA provides 6 types of metrics based on the format (`{lpar_name}_{value_type}_{lpar}`):

1. **_TOU_ Value type**

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

2. **_EFU_ Value type**

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

3. **_MSU_ Value Type**

Metric with **_MSU_** contains System's **MSU** value E.g. **RPRT_MSU_VIRPT**.

4. **_VC_ Value Type**

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

5. `_CHANNEL_` Value Type

Metrics with `_CHANNEL_` contains Channel's **CHACTUVC (Total utilization percent)** value. E.g. `RPRT_CHANNEL_11_FC`

6. `_JOB_` Value Type

Metrics with `_JOB_` contains **JUSPCPUD (CPU time for interval [in seconds])** value. E.g. `DVLP_JOB_ZOWESVR3`

App Queries

The ZEBRA API Endpoint has been restructured as follows:

`/ {api_version} / {lpar_name} / {rmf_monitor} / {report} ? {parameter} = {value}`

- **api_version**: represent the version of ZEBRA in use. For now, ZEBRA has only one api version. i.e. v1
- **lpar_name**: represent the lpar from which to retrieve RMF data. This name must be similar to lpar name (key) in dds.js file in the config directory.
- **rmf_monitor**: represent the monitor from which to retrieve data. ZEBRA accept the following values for rmf_monitor:
 - a. **rmf3** for monitor III
 - b. **rmfpp** for monitor I (post-processor)
 - c. **rmf** for RMF specific metrics(Reserved for RMF listmetrics)
- **report**: represent the report to retrieve from RMF. E.g CPC, PROC, CHANNEL, IOQ, WLMGL, CPU
- **parameter**: this represent RMF endpoint parameters like date, resource or Id.
- **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
- **lpar_parms (optional/filter)**
 - This is a parameter for 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.
- **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.

- **Start (start date [needed])**
This takes the start date of the report to retrieve from DDS. The date is in the format (YYYY-MM-DD). E.g start=2021-07-19
- **end (end date [needed])**
This takes the end date of the report to retrieve from DDS. The date is in the format (YYYY-MM-DD). E.g end=2021-07-19
- **SvcCls (optional/filter)**
This parameter takes the name of the service class for which to retrieve information about. E.g STCHIGH
- **Wlkd (optional/filter)**
This parameter takes the workload class name for which to retrieve information about. E.g TSO
- **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
- **Time (optional/filter)**
This parameter takes the exact time for which to retrieve information. The time is in the format HH.MM.SS

Examples:

Note: users can specify a different resource value to use when making request through ZEBRA by specifying a value for the resource Parameter. E.g.

<http://localhost:3090/v1/RPRT/rmf3/syssum?resource=,SYSPLEX>

If resource parameter is not provided in the request, ZEBRA will make use of the resource parameter in the Zconfig.json file.

- **RMF III Endpoints(CPC report)**
 - <http://localhost:3090/v1/RPRT/rmf3/CPC>
 - <http://localhost:3090/v1/RPRT/rmf3/CPC?parm=CPCHCMSU>
 - http://localhost:3090/v1/RPRT/rmf3/CPC?lpar_parms=VIRPT
 - http://localhost:3090/v1/RPRT/rmf3/cpc?lpar_parms=VIRPT&parm=CPCHLMS
U
- **RMF III Endpoints(USAGE and PROC report)**
 - <http://localhost:3090/v1/RPRT/rmf3/proc>
 - <http://localhost:3090/v1/RPRT/rmf3/proc?job=SDSFAUX>
 - <http://localhost:3090/v1/RPRT/rmf3/proc?parm=PRCPSVCL>
 - <http://localhost:3090/v1/RPRT/rmf3/proc?parm=PRCPSVCL&job=SDSFAUX>
Similar to USAGE Report
- **RMF I Endpoints (Workload Report)**
 - <http://localhost:3090/v1/RPRT/rmfpp/wlmgl?start=2021-07-17&end=2021-07-17>

OMP Mentorship Program 2020: Project ZEBRA

- <http://localhost:3090/v1/RPRT/rmfpp/wlmgl?start=2021-07-17&end=2021-07-17&SvcCls=STCHIGH>
- <http://localhost:3090/v1/RPRT/rmfpp/wlmgl?start=2021-07-17&end=2021-07-17&SvcCls=STCHIGH&Time=05.30.00>
- <http://localhost:3090/v1/RPRT/rmfpp/wlmgl?start=2021-07-17&end=2021-07-17&SvcCls=STCHIGH&duration=05.30.00,09.30.00>
- <http://localhost:3090/v1/RPRT/rmfpp/wlmgl?start=2021-07-17&end=2021-07-17&Wlkd=TSO>
- <http://localhost:3090/v1/RPRT/rmfpp/wlmgl?start=2021-07-17&end=2021-07-17&Wlkd=TSO&Time=05.30.00>
- <http://localhost:3090/v1/RPRT/rmfpp/wlmgl?start=2021-07-17&end=2021-07-17&Wlkd=TSO&duration=04.00.00,07.30.00>
- **RMF I Endpoints (CPU Report)**
- <http://localhost:3090/v1/RPRT/rmfpp/cpu?start=2021-07-17&end=2021-07-17>
- **RMF Specific Fields**
- <http://localhost:3090/v1/RPRT/rmf?id=8D0F60>
- <http://localhost:3090/v1/RPRT/rmf?id=8D0160&resource=„SYSPLEX>
- **Listing IDs and Description**
- <http://localhost:3090/v1/RPRT/rmf?id=LIST>
- <http://localhost:3090/v1/RPRT/rmf?id=LIST&resource=„SYSPLEX>
- **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.

 - a. <http://localhost:3090/static?file=C:\Users\Salis\Desktop\rmfpp.xml>

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

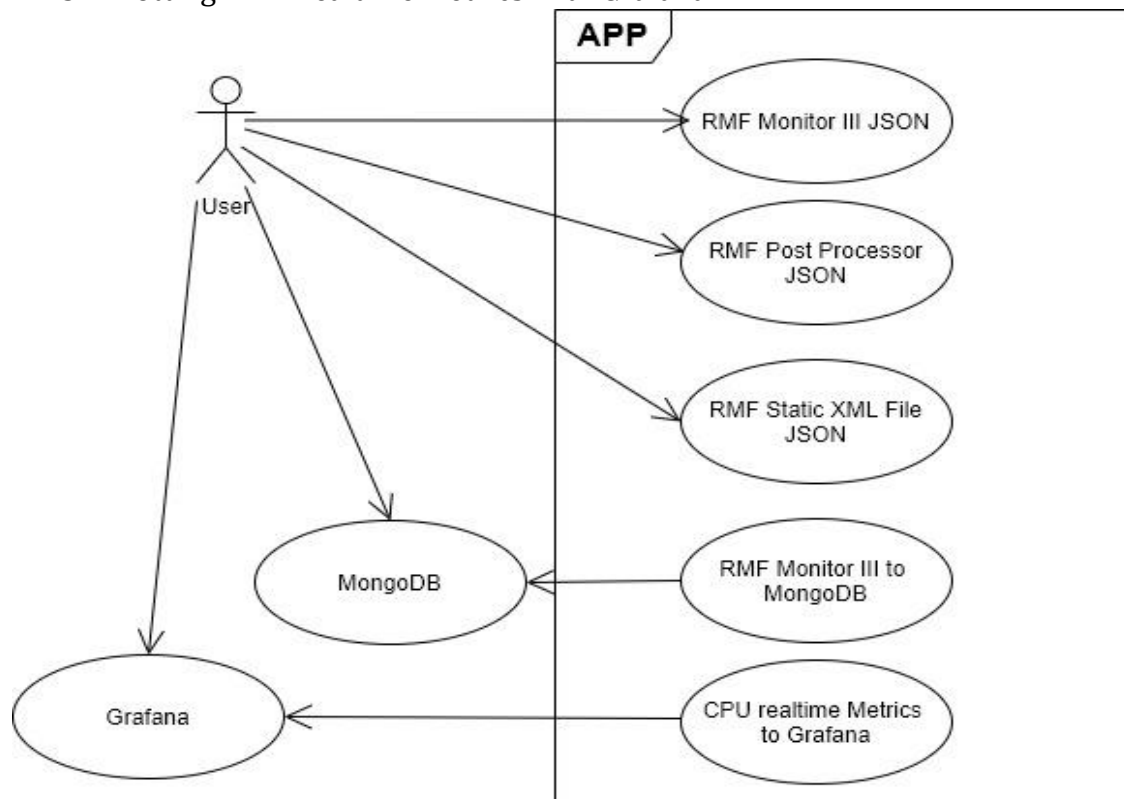


Figure 1: Use Case Diagram

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.

- **promMetricsv1.js**

This file contains function that expose real time CPU utilization metrics using prom-client library.

- **Mongov1.js**

This file contains functions that save real time parsing engine output to MongoDB.

ii. App Server Directory

This directory Consist of:

- **V1_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 contains 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/Error Pages.

Architecture

- 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.

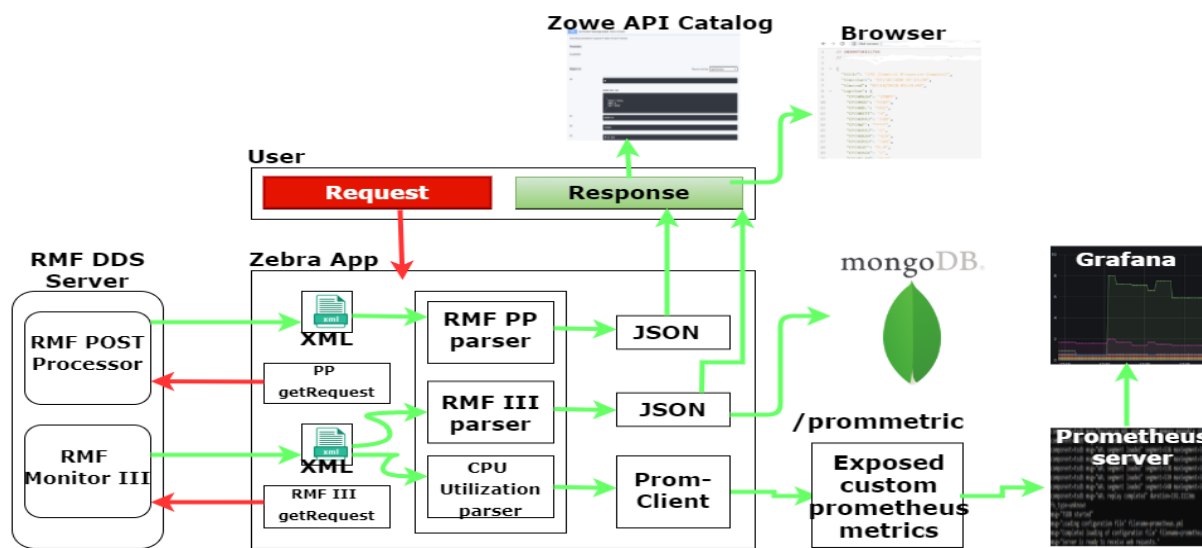


Figure 2: Project Architecture

- 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

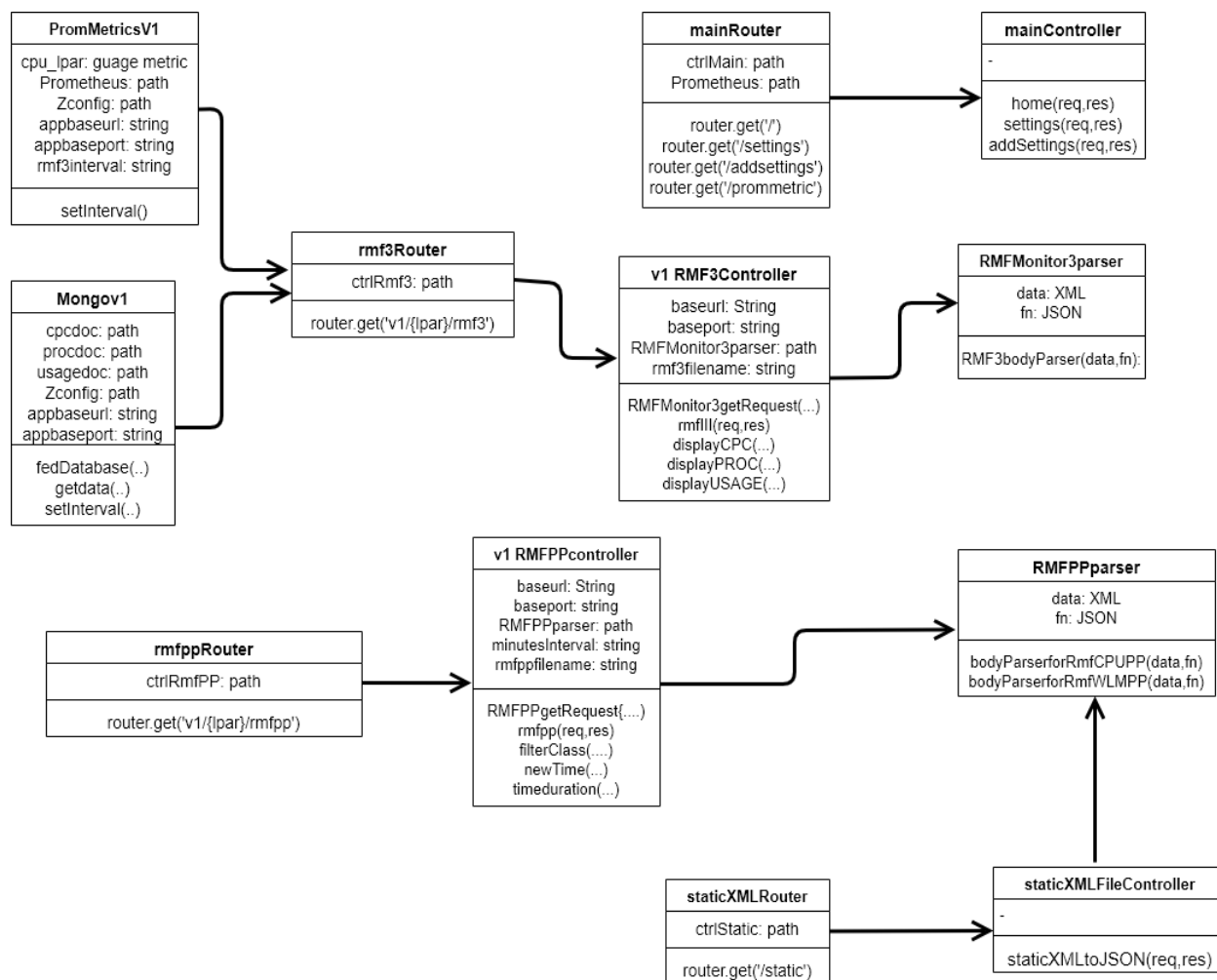


Figure 3: Project Class Diagram

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.

Mongov1 and **PromMetricsv1** 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 v1router 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 **PromMetricsv1** 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

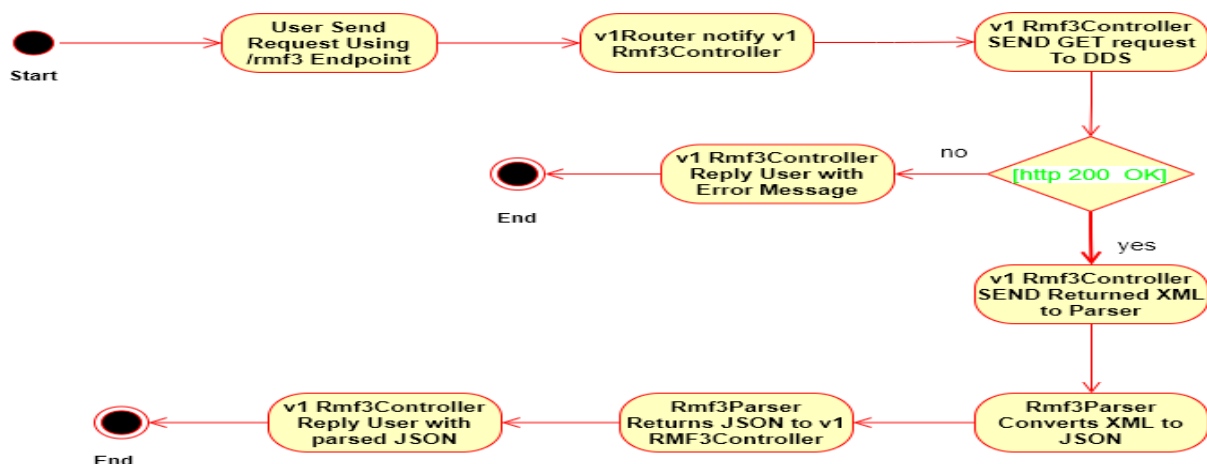


Figure 4: RMF 3 Activity Diagram

RMF Monitor III Activities starts when a User trigger a request using the /v1/{lpar}/rmf3 Endpoint, This leads to a series of activities involving v1Router, 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

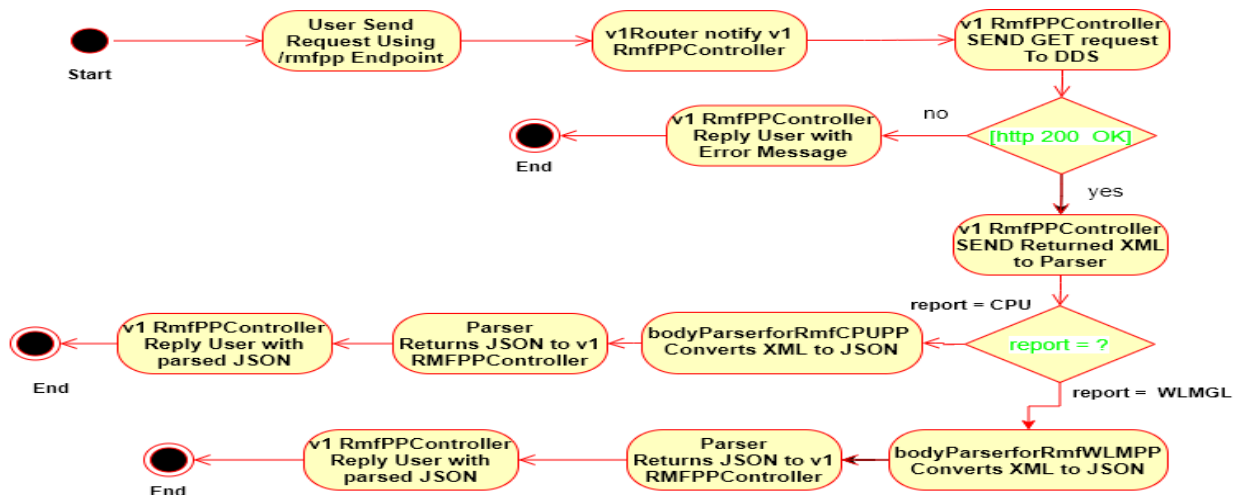


Figure 5: RMF Post Processors Activity Diagram

RMF Monitor I Activities starts when a User trigger a request using the /v1/{lpar}/rmfpp Endpoint, This leads to a series of activities involving v1Router, 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.

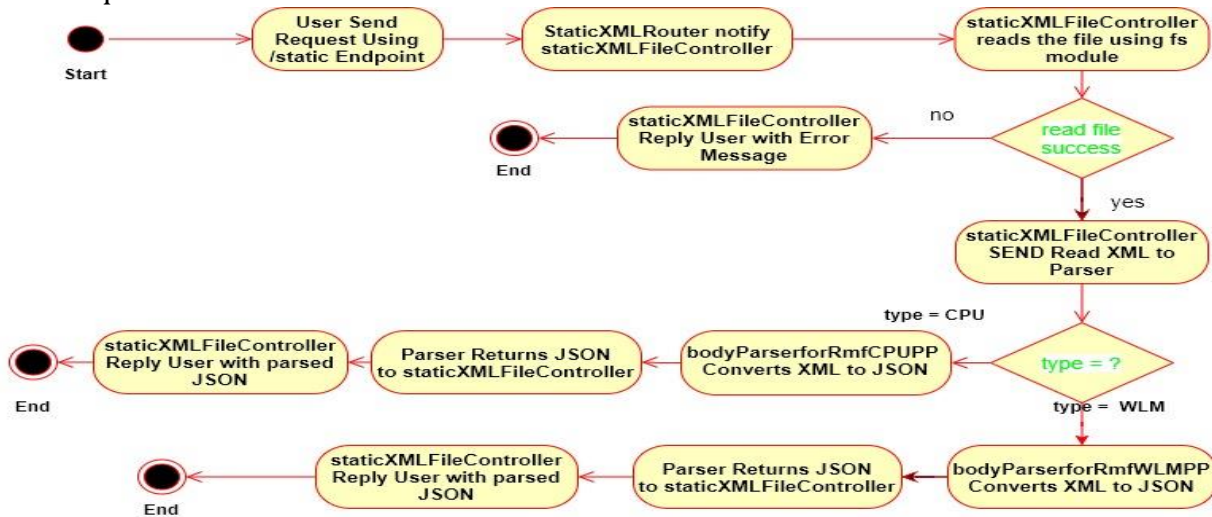


Figure 6: Static XML File Activity Diagram

iv. Mongo Activity Diagram

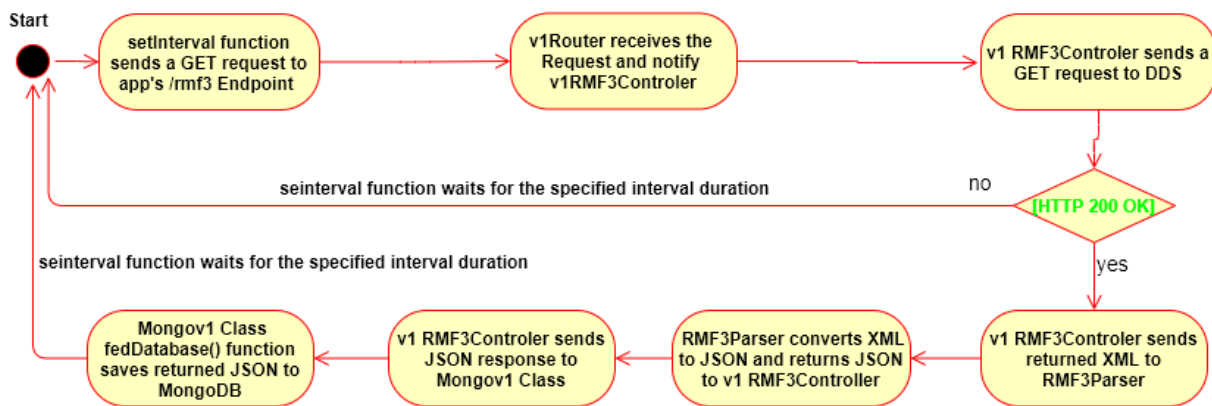


Figure 7: Mongo Activity Diagram

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. Prometheus Realtime Metrics Activity Diagram

Just like in Mongo Class, Activities of the prometheus Realtime Metrics (PromMetricsv1) class do not need user intervention. The setInterval function triggers RMF III Activities which returns a JSON. Through a call back function, Prometheus 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.

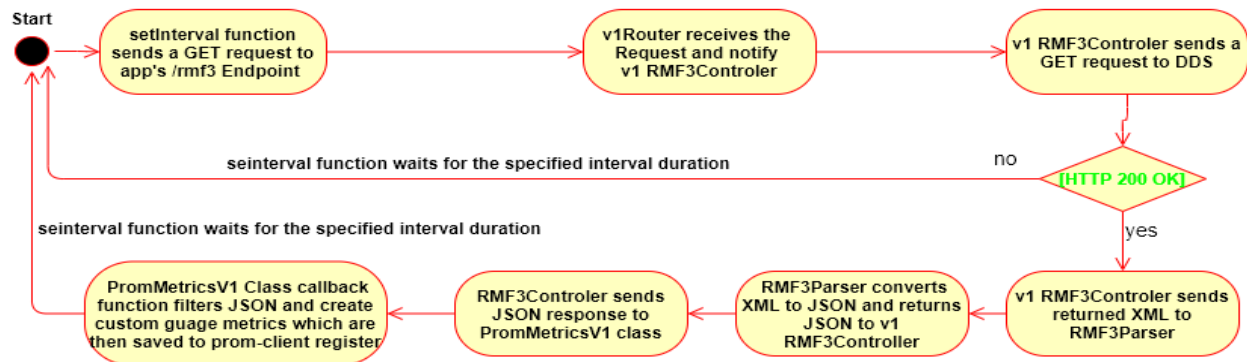


Figure 8: Prometheus Realtime Metrics Activity Diagram

Sequence Diagram

i. RMF Monitor III Sequence Diagram

RMF monitor III Service is triggered when user send a request using /rmf3 Endpoint (e.g. localhost:3000/v1/{lpar}/rmf3/CPC).

V1Router 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.

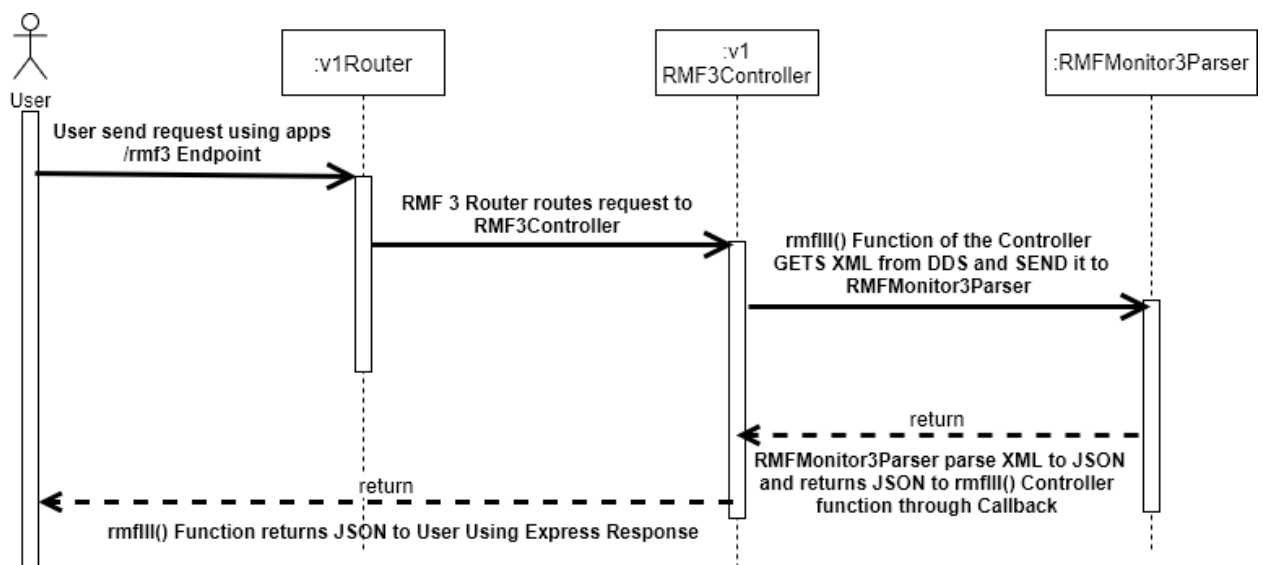


Figure 9: RMF 3 Sequence Diagram

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/v1/{lpar}/rmfpp/CPU).

V1Router 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.

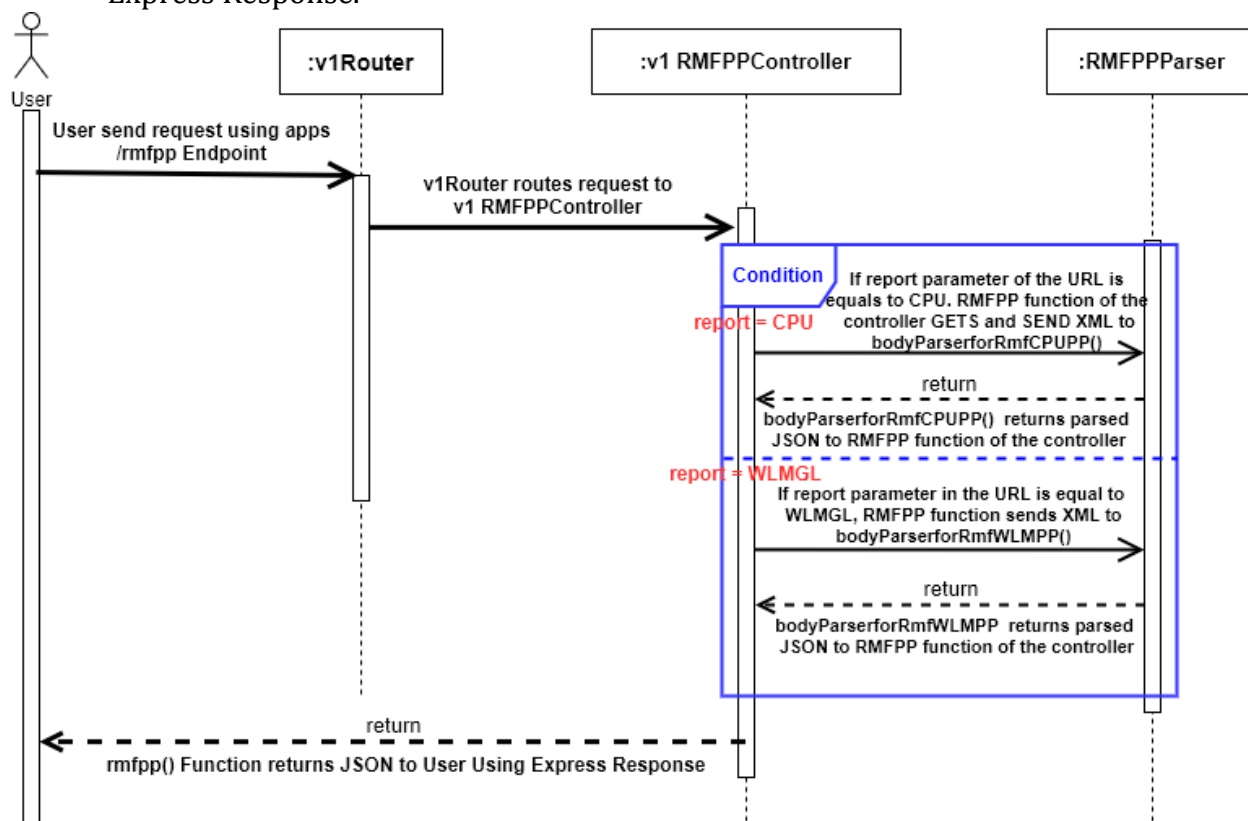


Figure 10: RMF PP Sequence Diagram

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 rmfpparser 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.

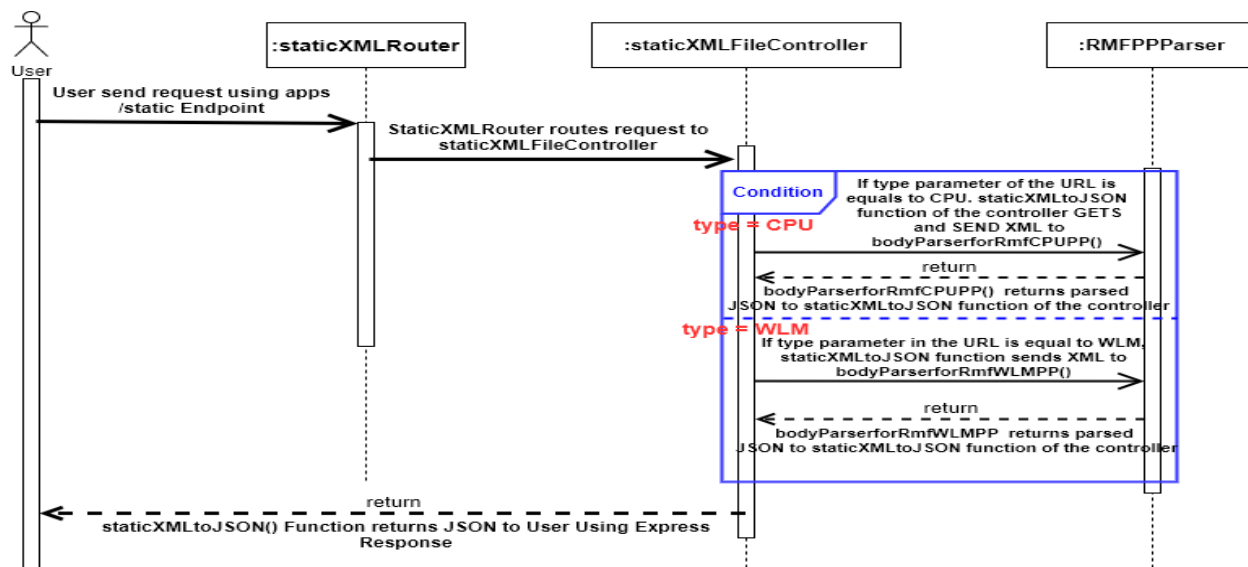


Figure 11: Static XML File Sequence Diagram

iv. Mongo

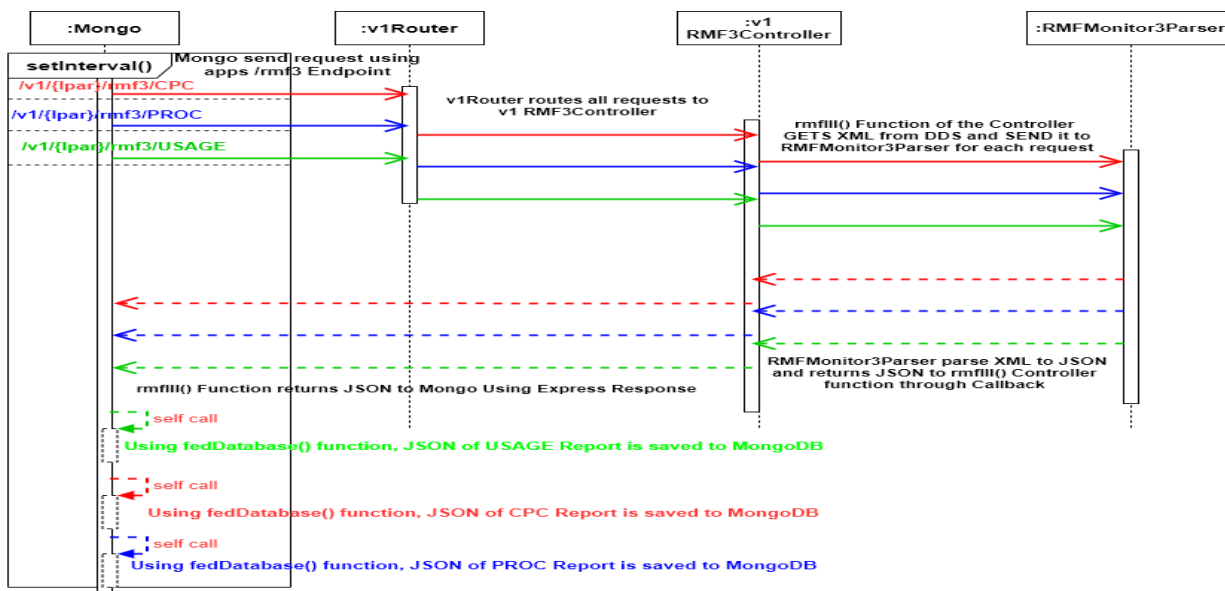


Figure 12: Mongo Sequence Diagram

Unlike the first three services, Mongov1 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 /rmf3 endpoint. For each report type (CPC, PROC And USAGE), v1Router 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 Mongov1 class.

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

v. Prometheus Realtime Metrics

The Prometheus 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.

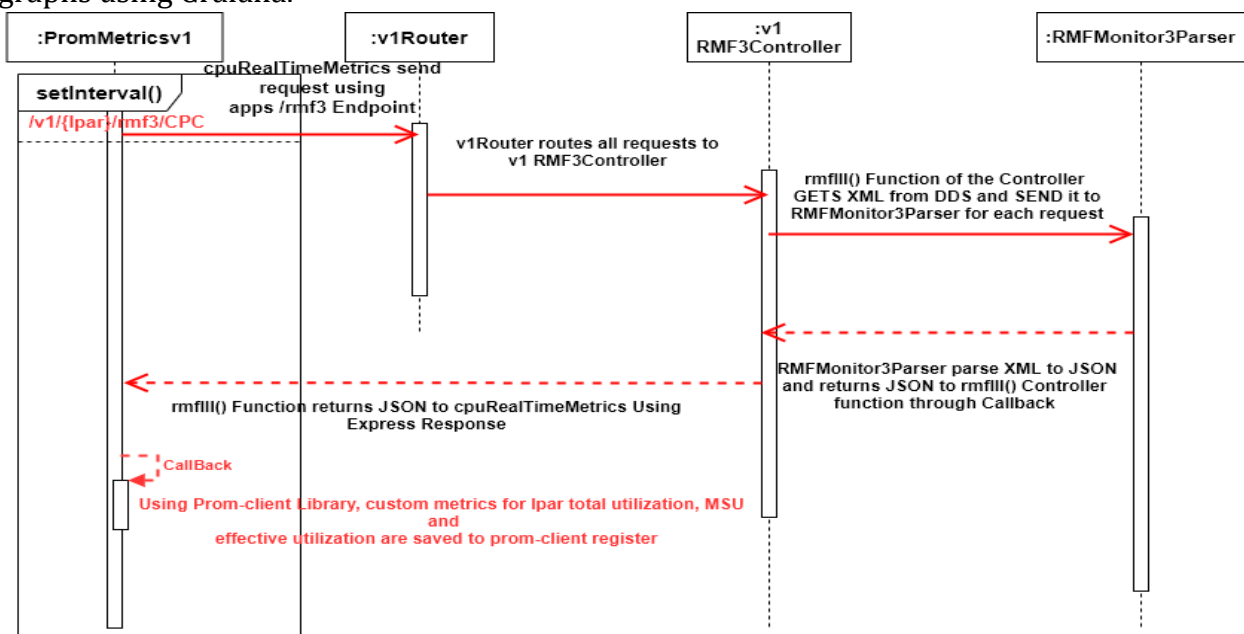


Figure 13: CPU Realtime Metrics Sequence Diagram

The setInterval function of the CPU real-time metrics class sends a request to the app /rmf3 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

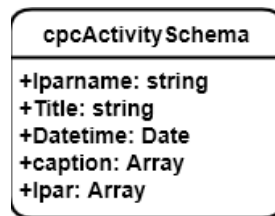


Figure 14: CPC Schema Definition

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

- **Lpar_name**
This represent the name of the lpar from which the report was retrieved.
 - **title:**
Title is a string that represent the report title
 - **DateTime**
This is a datetime object that represent the CPC report **date and time**.
 - **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 data as a series of documents.

2. PROC Activity

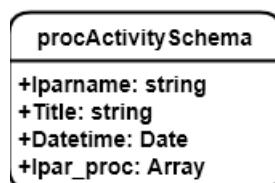


Figure 15: PROC Schema Definition

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

- **Lpar_name**
This represent the name of the lpar from which the report was retrieved.
- **title:**
Title is a string that represent the report title
- **DateTime**
This is a datetime object that represent the CPC report **date and time**.
- **Lpar_proc**
This is an array and is populated by an array of PROC report **table** key

The Mongo Class Create PROC Activity data as a series of documents.

3. USAGE Activity

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

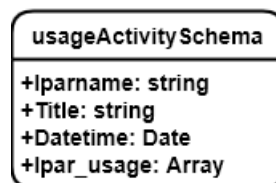


Figure 16: Usage Schema Definition

- **Lpar_name**
This represent the name of the lpar from which the report was retrieved.
 - **title:**
Title is a string that represent the report title
 - **DateTime**
This is a datetime object that represent the CPC report **date and time**.
 - **Lpar_usage**
This is an array and is populated by an array of USAGE report **table** key
- The Mongo Class Create USAGE Activity data as a series of documents.

4. Workload Activity

For USAGE report, the main document definition is named workloadActivitySchema. The document contains the following fields:

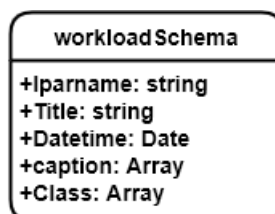


Figure 16: Workload Schema Definition

- **Lpar_name**
This represent the name of the lpar from which the report was retrieved.
- **title:**
Title is a string that represent the report title
- **DateTime**
This is a datetime object that represent the CPC report **date and time**.
- **Caption**
This is an array and is populated by Workload report **caption** key.
- **Class**
This is an array and is populated by an array of the workload report **Classes**.