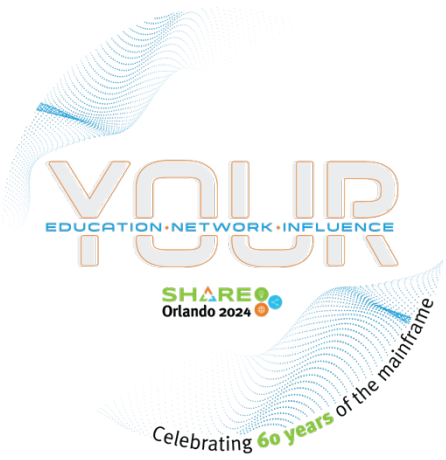


LAB:
Z/OSMF – CHOOSE YOUR OWN TOPIC

Getting started with
IBM SMF Explorer

Alexander Giemsa
IBM Corporation

Alexander.giemsa@ibm.com

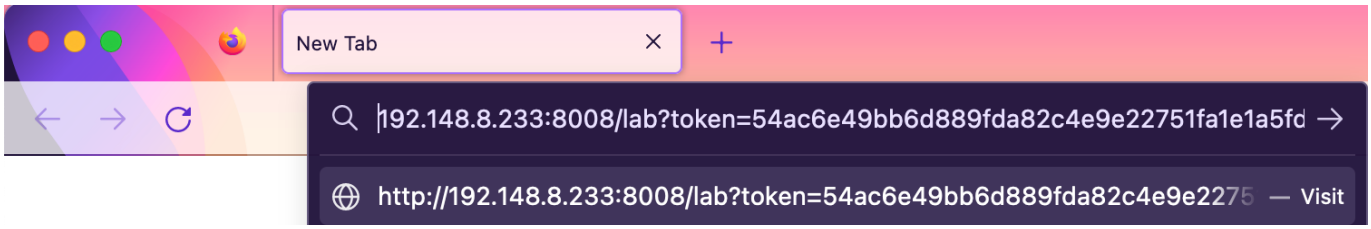
Introduction

IBM® SMF Explorer is a Python library that simplifies the accessing and processing of SMF data. SMF Explorer can be used in a Jupyter Lab environment.

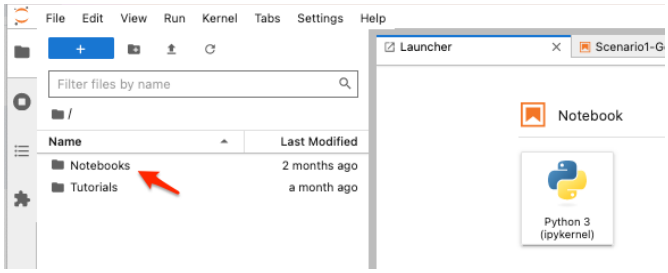
Usually, you install IBM SMF Explorer on your workstation or on a separate server on top of the Python server. To simplify this lab, we created a Docker image that can be started on Linux on Z or zCX environment. For this lab, we are using a zCX instance started on SHARE LPAR. That way, instead of installing and starting SMF Explorer on your workstation, we will start your own SMF Explorer Docker container on zCX running on z/OS.

Introduction Notebook – Performance Analysis

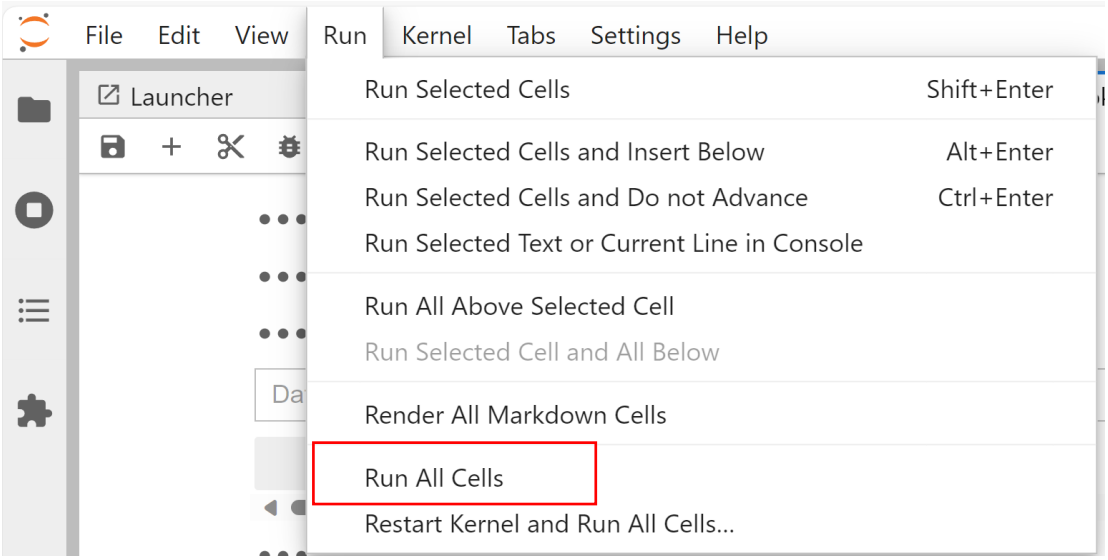
1. Normally, you need to start the Jupyter Lab session from the command line. For the sake of time, the session is started for you in advance. You will get the link to the Jupyter Lab from the instructor. Copy the URL, open the browser (for example Firefox) and paste the link that contains the address of your JupyterLab into it:



2. These actions open a JupyterLab session in your browser. You can find a folder "Notebooks/Scenarios" containing the Jupyter Notebooks, which are used in this lab.



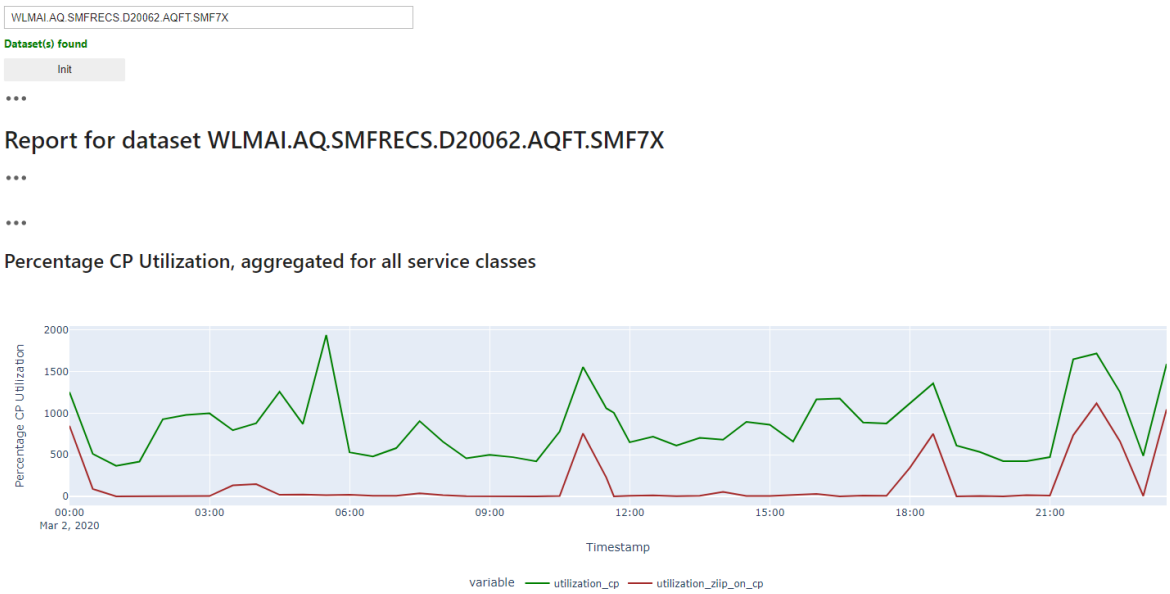
3. Double-click on Notebooks folder and then go to Scenarios folder and double-click *Introduction Notebook - Performance Analysis.ipynb* to open the Introduction Notebook. This notebook contains code to fetch SMF data using SMF Explorer, then visualize it for further analysis. You will learn how to create such notebooks during the course of this lab. But first, let's see how such report notebook looks like.
4. To run all cells of the notebook at the same time, click Run > Run all Cells:



5. This will open a dialog box where you can specify the name of the SMF data set to fetch the data from. Enter *PMUENCH.SMFDATA.WLMAI.SMF7X.SHARB{xx}* in the box and press *Init*.
Note: *xx* is given to you by the instructor.

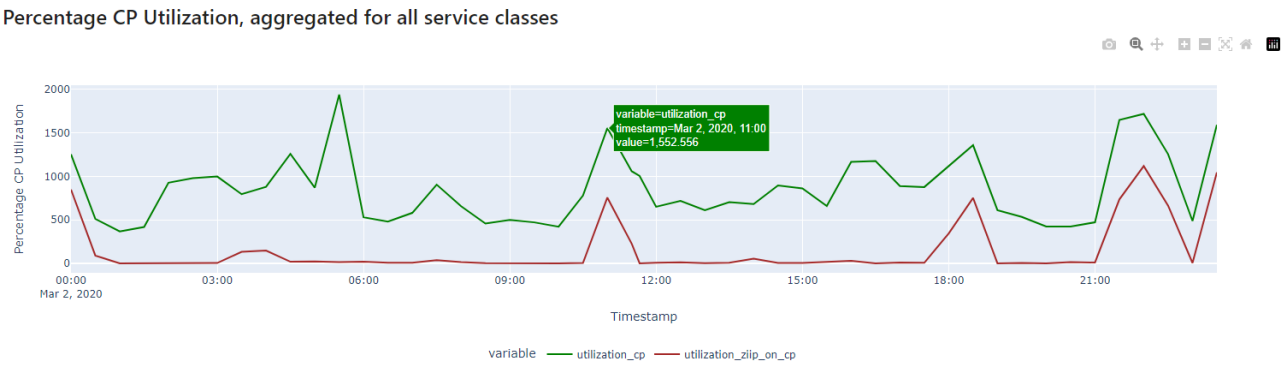
It will then take several minutes for the notebook to fetch, prepare and visualize the data.

6. The first generated graph demonstrates the percentage CP utilization over a course of 24 hours. The green line is the percentage CP Utilization. The red line is the percentage CP Utilization of zIIP eligible workload.

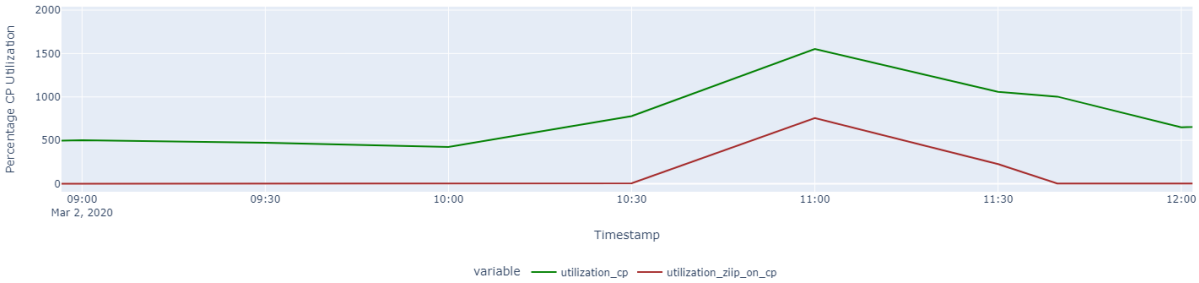


This chart is interactive, which means you can zoom in, zoom out, activate, and deactivate chart components to gain insight. You can hover over a specific point in the chart to see the data values at that time point. Let's try this out!

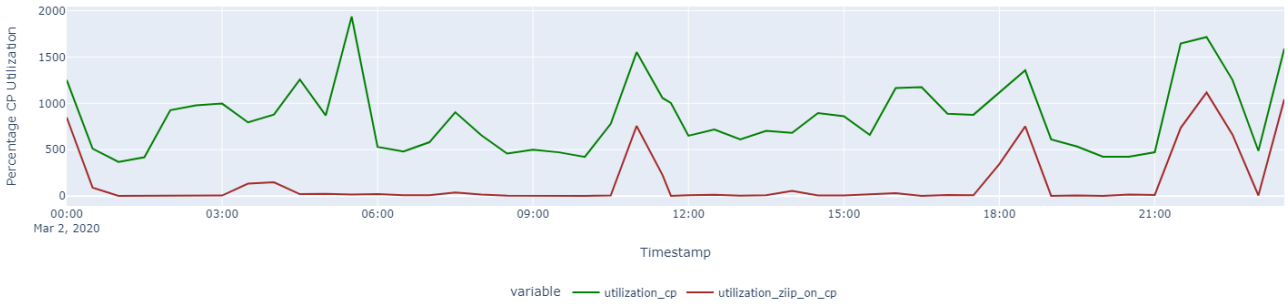
a. Hover the mouse over the *utilization_cp* line at 11:00 to see the CP utilization at that time point:



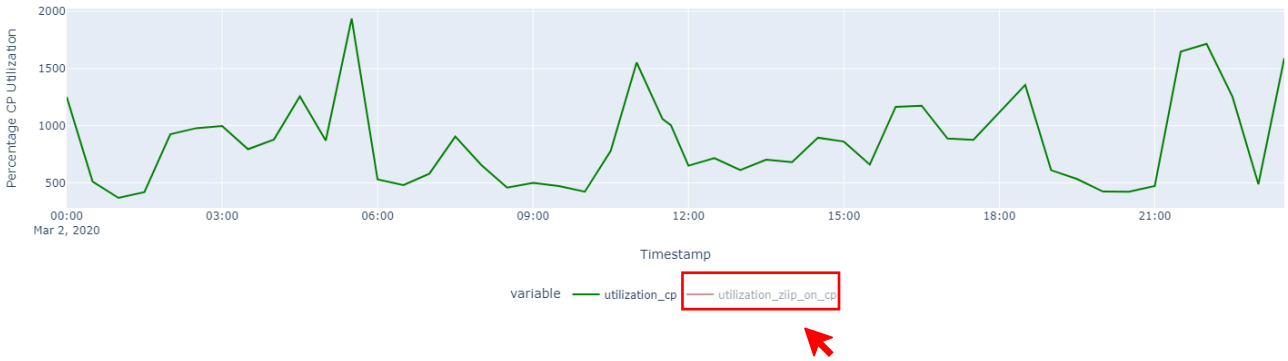
b. Use the mouse to drag the area between 09:00 and 12:00 for a closer view of the chart:



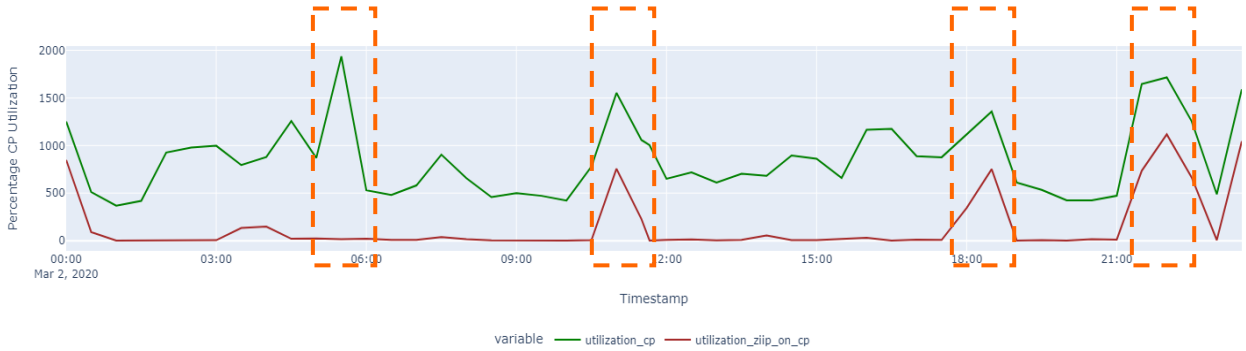
c. Reset the chart by double clicking anywhere in the chart:



d. You can deactivate a line in the graph by clicking on the name of the line in the legend. For example, click on *utilization_ziip_on_cp* to hide this line from the graph. Then click on it again to show the line:

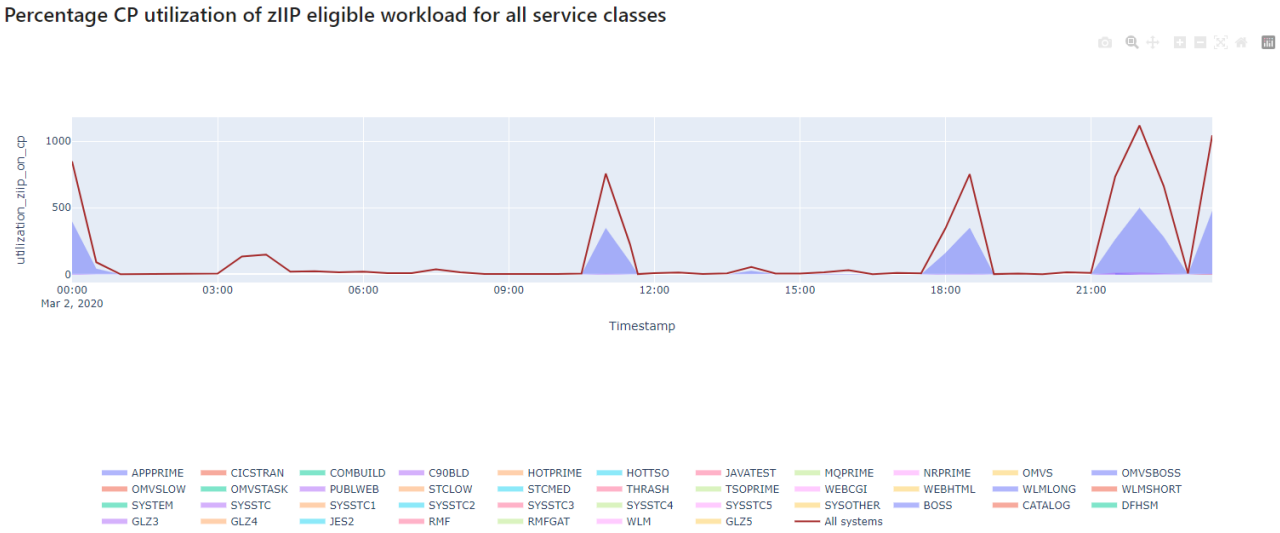


As you may notice, the graph for CP Utilization demonstrates distinct peaks at time point 05:30, 11:00, 18:30 and 22:00:

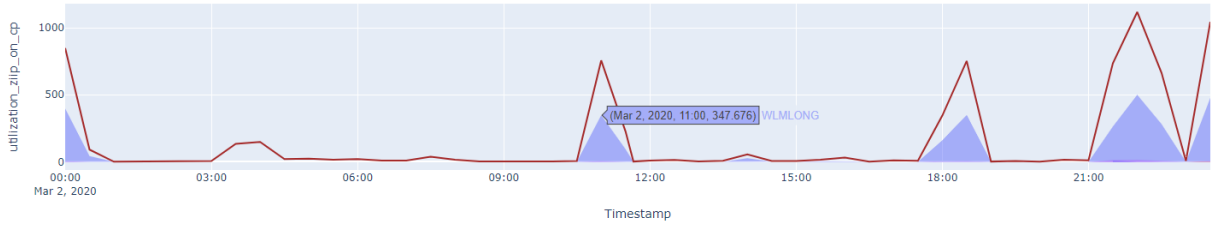


From the chart, it is evident that the last 3 peaks are caused by *utilization_ziip_on_cp,* which are zIIP eligible jobs that run on CP.

7. Let's look which service class has the highest *utilization_ziip_on_cp* in the second chart*:*

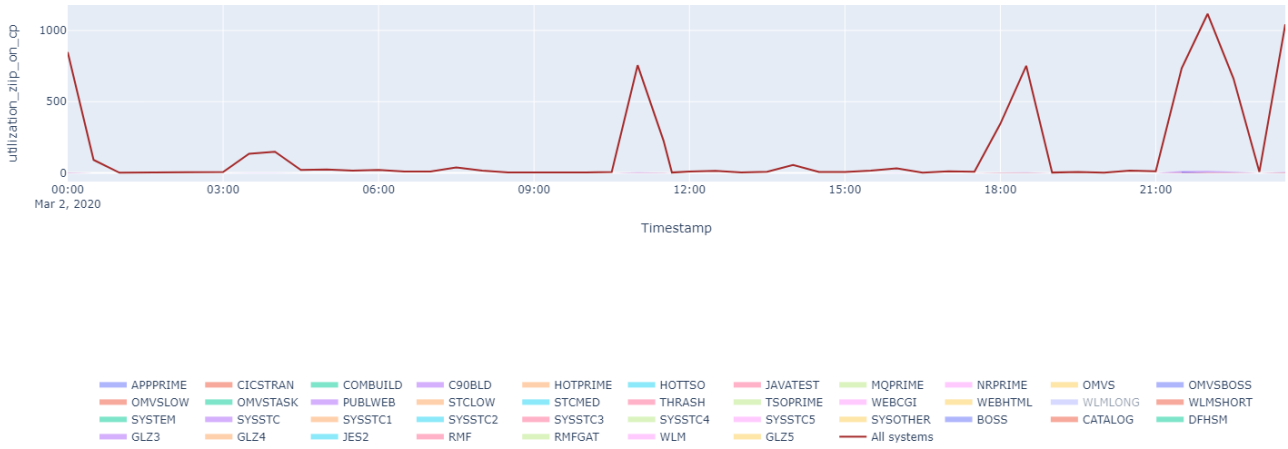


The red line is the *utilization_ziip_on_cp* that is summed over all service classes. The *utilization_ziip_on_cp* of each service class is displayed as a filled area. If you hover over the purple area, you can see that this area belongs to the service class WLMLONG:



That suggests that the service class WLMLONG contributes the most to the total *utilization_ziip_on_cp.*

8. We can verify this assumption by hiding the visualization for service class WLMLONG:



As you can see, the *utilization_ziip_on_cp* of the other service classes is very low compared to the one of WLMLONG. This verifies our assumption that WLMLONG contributes the most to the total *utilization_ziip_on_cp*. This also suggests that the peaks in the *utilization_ziip_on_cp* are mostly caused by the service class WLMLONG.

9. In the next part of the notebook, you will find a toggle button that list all service classes. When you select one service class, the following visualizations will be updated accordingly.
10. Next, select the class WLMLONG to activate the description and visualization for this service class.

The third graph demonstrates the utilization of several types of processors:

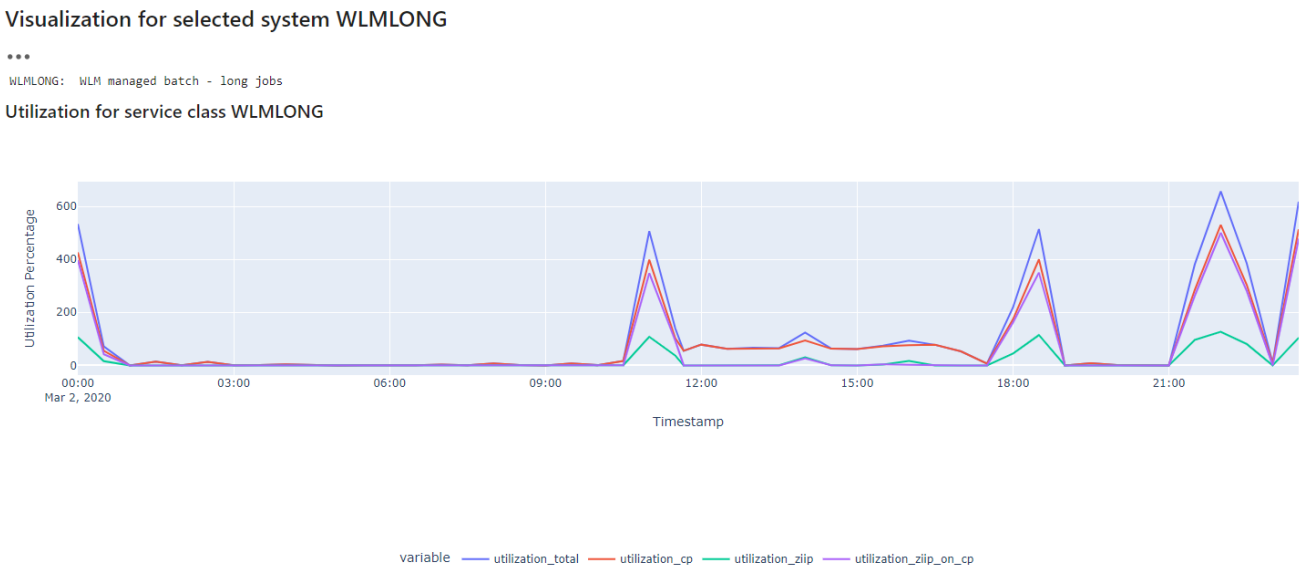
utilization_total: Percentage application utilization, summing CPU, SRB, RCT, I/O interrupt and hyperspace times.

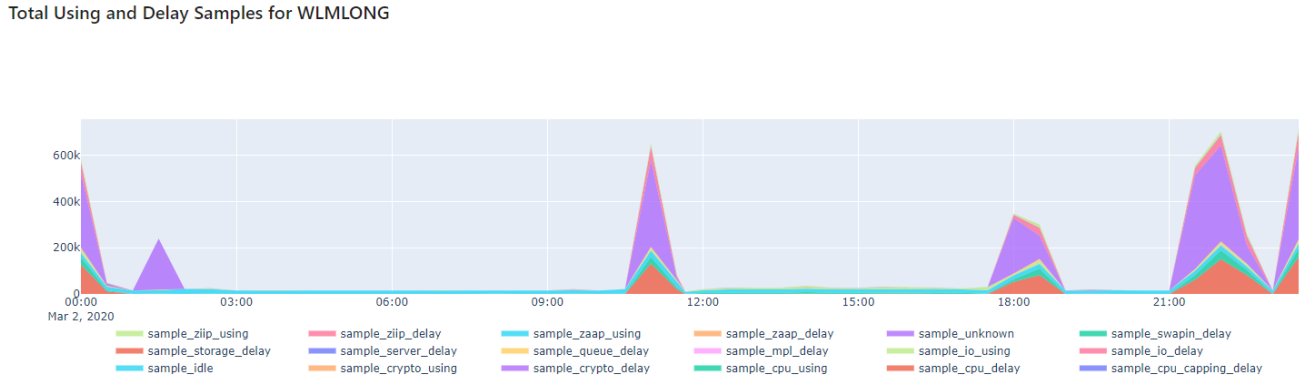
utilization_cp: Percentage CPU utilization, i.e. application utilization minus zIIP and zAAP utilization.

utilization_ziip: Percentage zIIP utilization

utilization_ziip_on_cp: Percentage CP utilization of zIIP eligible workload.

From the graph, it is evident that the peaks in the *utilization_ziip_on_cp* of the service class WLMLONG cause peaks in *utilization_total.* This suggests that there are not enough available zIIP processors.





11. You can also try visualizing other service classes by choosing the service class name in the toggle button. You will see that the serive class description as well as the last two graphs will be updated automatically on selection.

Conclusion:

This notebook demonstrates the combination of SMF Explorer with other Python libraries to create meaningful notebooks to analyze your system requirements. From the notebook, we can see that most of the peaks in the utilization of the CP processors are caused by zIIP-eligible job. This suggests that in this test sytem, there are not enough available zIIP processors. Besides, the service class WLMLONG is the main cause for the high number of zIIP-elegible jobs that run on the CPU.

This was a simple demonstration of what can be done using SMF Explorer and other Python libraries on Jupyter Notebook. You can leverage your Python skills to create your own Jupyter Notebooks that serve your own needs.

Excited to try more? The exercises 1-5 shows you step by step how to work with SMF Explorer and create such notebook.

Exercise 1 – Get used to Jupyter Lab

Now as you know what can be done using Jupyter Notebook, let's take a closer look at how to create such insightful notebooks. Double-click on *Exercise 1 – Get used to Jupyter Lab* to open the Jupyter Notebook.

1. 1 Overview of Jupyter Notebook and Jupyter Lab

A Jupyter Notebook contains a set of cells that can hold either code, markdown text or raw text. A cell is the fundamental unit of organization within a notebook.

- You can enter and run Python code in a code cell. When you run a code cell, the code is run, and the output (if any) is displayed after the cell.
- Using a Markdown cell, you can write formatted text that uses Markdown syntax. Markdown is a lightweight markup language that you can use to add document or explanations to your code. With markdown, you can add contents such as headings, lists, links, images, tables, ...
- Raw cells hold content that is not to be modified or run. The contents of a raw cell are saved exactly as entered and displayed as-is without any processing.

On the other hand, Jupyter Lab is an environment where you can create and run Jupyter Notebook.

Code Cell

To run a code cell, you first select the cell that you want to run. You can run a cell in two different ways. The first way is to use the keyboard shortcut Shift + Enter to run the currently selected cell and move the focus to the next cell. Select the first cell and run with the keyboard shortcut:

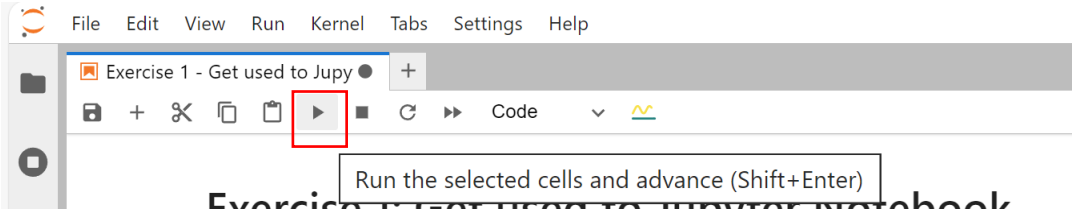
```
[1]: # This is a simple Python code cell
     print('Hello, SMF Explorer!')

     Hello, SMF Explorer!
```

The first line in the cell is a comment in Python. The second line is a code line that contains a print statement which output the text *Hello, SMF Explorer!* to the notebook.

When a code cell is successfully executed, a number will be shown to the left of the cell and the result of the code is shown directly. Note that sometimes you will see an asterisk (*) symbol instead of a number. This means that the code cell is still being executed. Wait until the execution completes.

Another way is to click the Run button in the toolbar to run the currently selected cell. Select the second code cell and click the Run button.

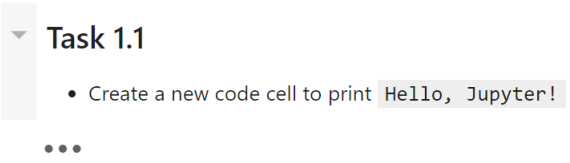


The result of this action is:

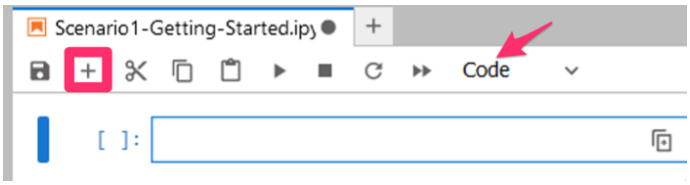
```
[2]: # Perform a basic mathematical operation
     result = 24 + 10
     result

[2]: 34
```

Each exercise contains several tasks. After finishing the task, you can click the 3 dots symbol to reveal the result. Let's start with the Task 1.1:



To create a new code cell, click the + button in the toolbar. Make sure that the type of Code for the newly created cell is selected:



Then type *print('Hello, Jupyter!')* into the code cell and execute it:

```
print('Hello, Jupyter!')

Hello, Jupyter!
```

Markdown Cell

The next cell is a markdown cell that contains formatted text.

```
# Hello, SMF Explorer!
## Hello, SMF Explorer!

**Hello, SMF Explorer!**

Hello, SMF Explorer!
```

The first and the second line is the first and the second-level heading, accordingly. The double asterisks (**) in the third line are used to make the enclosed text bold. Lastly, the last line is just plain text, which is displayed as-is in the output.

After editing a Markdown cell, you use Shift + Enter to display the text. The result of this action is:

Hello, SMF Explorer!

Hello, SMF Explorer!

Hello, SMF Explorer!

Hello, SMF Explorer!

Conclusion

Now, as you know the basic of Jupyter Notebook, follow the instructions and exercises in our notebooks to get to know SMF Explorer. There are 5 exercises for you, each contains several sub tasks. We recommend you to follow the exercises one by one.