SMF analysis using Apache Spark and Jupyterlab

April 10th 2024 – GSE z/OS Expertenforum

Marcel Schmidt

Agenda

- Introduction
- Infrastructure overview
- Component description
- Building the infrastructure
  - WSL (Windows Subsystem for Linux)
  - Apache Spark
  - Jupyterlab
  - PostgreSQL
  - Nvidia CUDA (Compute unified device Architecture)
  - Tesla T4 accelerator card
- Demo

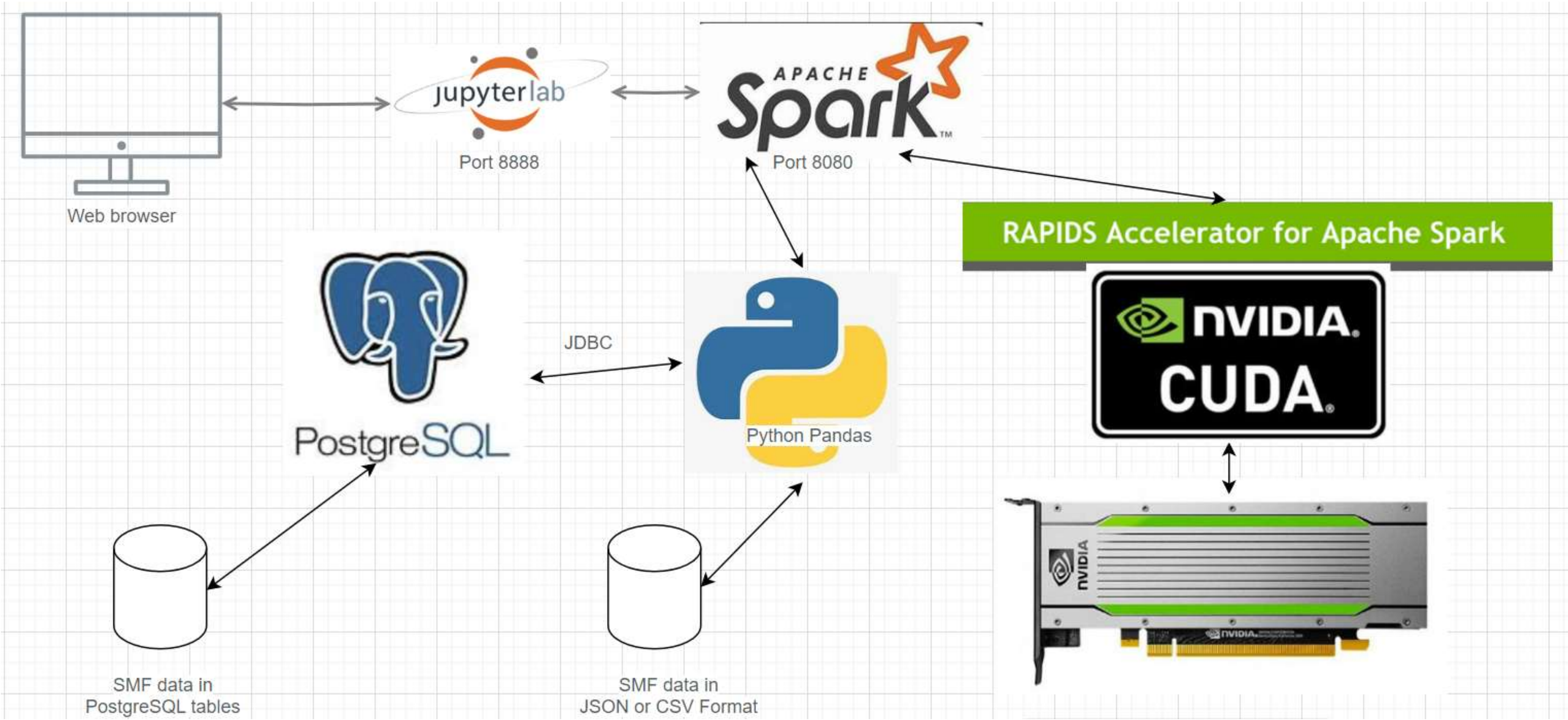

INTRODUCTION

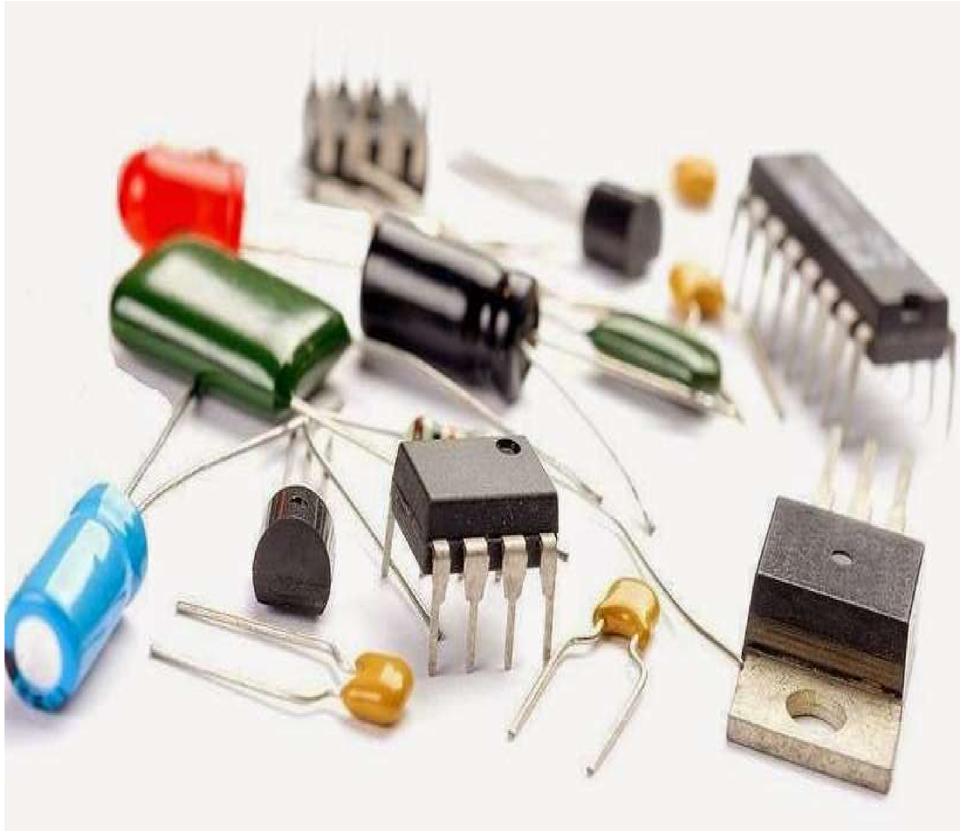
Introduction

- It is possible to analyze SMF data using an assembly of open source technologies
- The necessary infrastructure can be built on a Windows or Unix platform
- The SMF records must be transformed and made available on the analysis platform either as JSON or CSV files or PostgreSQL records
- There are commercial solutions available that are combining these open source tools with proprietary code to directly access SMF data residing on z/OS

Infrastructure Overview

Infrastructure overview



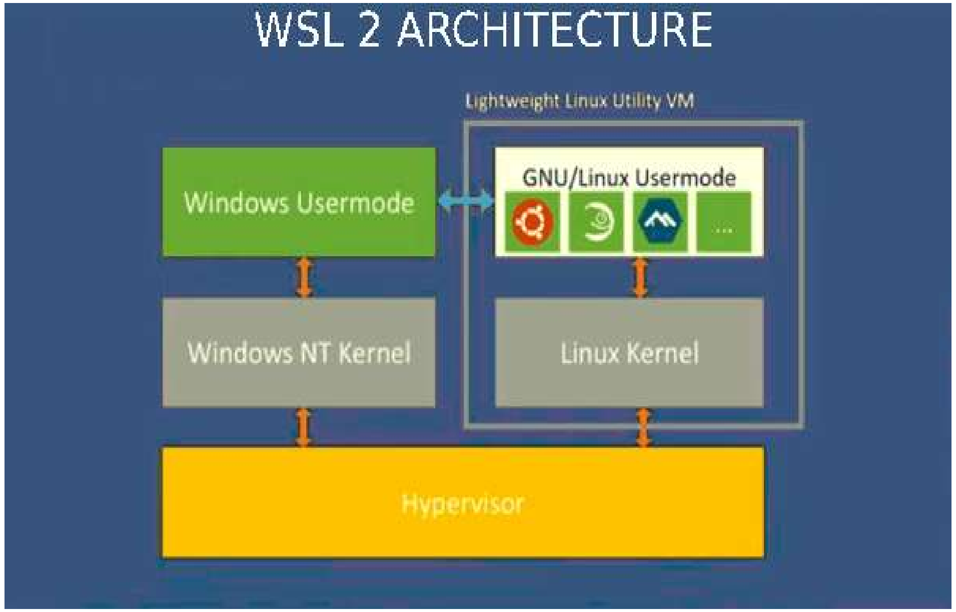


Component description

Component description - WSL2

- Windows Subsystem for Linux V2

Allows you to run a Linux environment on a Windows machine without the need for a separate virtualization solution.



Component description – Apache Spark

- Unified analytics engine for large-scale dataprocessing.
Aka "Big Data" and "Hadoop"

- Spark Core provides distributed task dispatching, scheduling, and basic I/O functionalities, exposed through an API for Java, Python, Scala, .NET and R

- Spark SQL is Apache Sparks module for working with structured data that is abstracted as «data frames»

- A Data frame is a two-dimensional data structure, i.e., data is aligned in a tabular fashion in rows and columns.
Pandas DataFrame consists of three principal components, the data, rows, and columns.

Component description – Apache Spark (2)

- Uniform data access - Connect to any data source the same way.

- DataFrames and SQL provide a common way to access a variety of data sources, including Hive, Avro, Parquet, ORC, JSON, and JDBC. You can even join data across these sources.

- Example:

```
spark.read.json("s3n://...")
.registerTempTable("json")
results = spark.sql("""SELECT * FROM people JOIN json ...""")
```

Component description – Jupyterlab

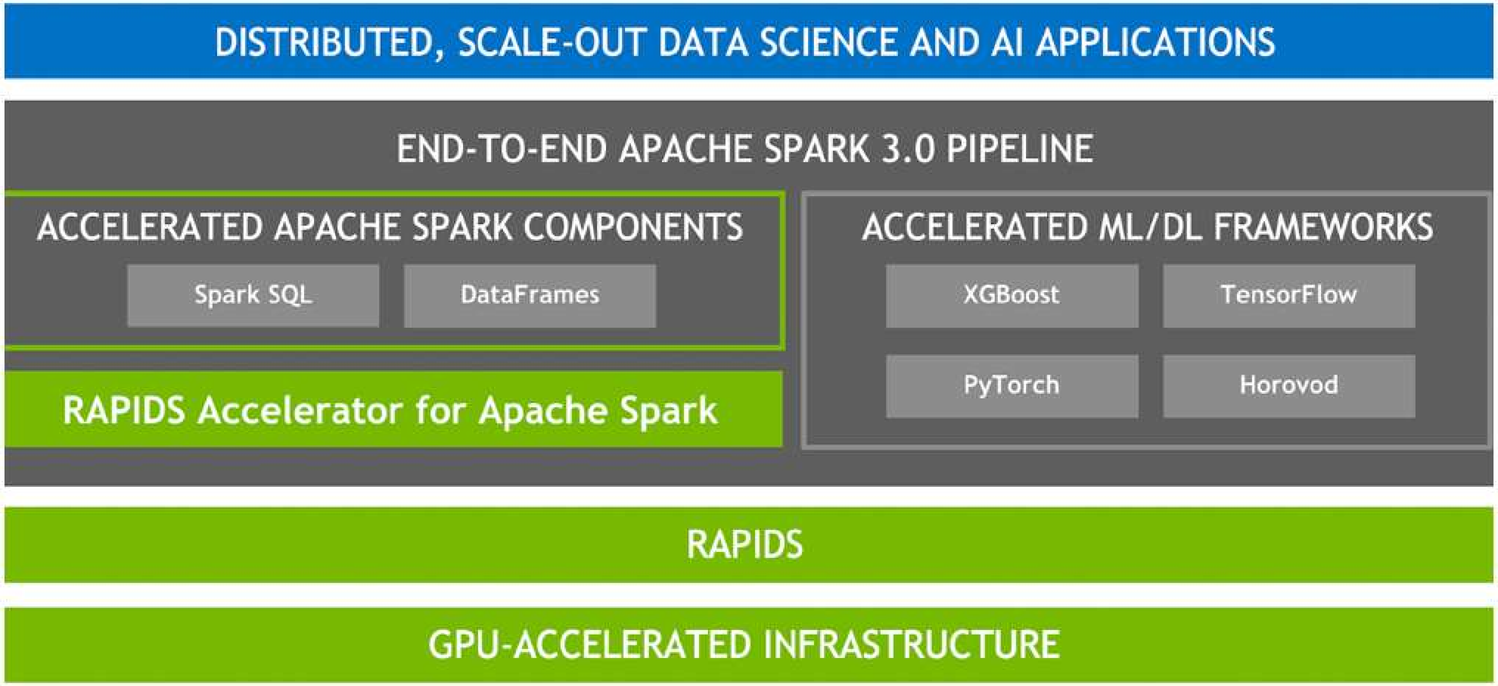
- web-based interactive development environment for notebooks, code, and data.
- Its flexible interface allows users to configure and arrange workflows in data science, scientific computing, computational journalism, and machine learning.
- The JupyterLab environments provide a productivity-focused redesign of Jupyter Notebook. It introduces tools such as a built-in HTML viewer and CSV viewer along with features that unify several discrete features of Jupyter Notebooks onto the same screen.

Component description – PostgreSQL

PostgreSQL is a powerful, open source object-relational database system with over 35 years of active development that has earned it a strong reputation for reliability, feature robustness, and performance.

There is a wealth of information to be found describing how to install and use PostgreSQL through the official documentation.

Component description – RAPIDS Accelerator



RAPIDS Accelerator for Apache Spark Release v21.10 | NVIDIA Technical Blog

Component description – Nvidia CUDA

CUDA® is a parallel computing platform and programming model developed by NVIDIA for general computing on graphical processing units (GPUs). With CUDA, developers are able to dramatically speed up computing applications by harnessing the power of GPUs.

In GPU-accelerated applications, the sequential part of the workload runs on the CPU – which is optimized for single-threaded performance – while the compute intensive portion of the application runs on thousands of GPU cores in parallel.

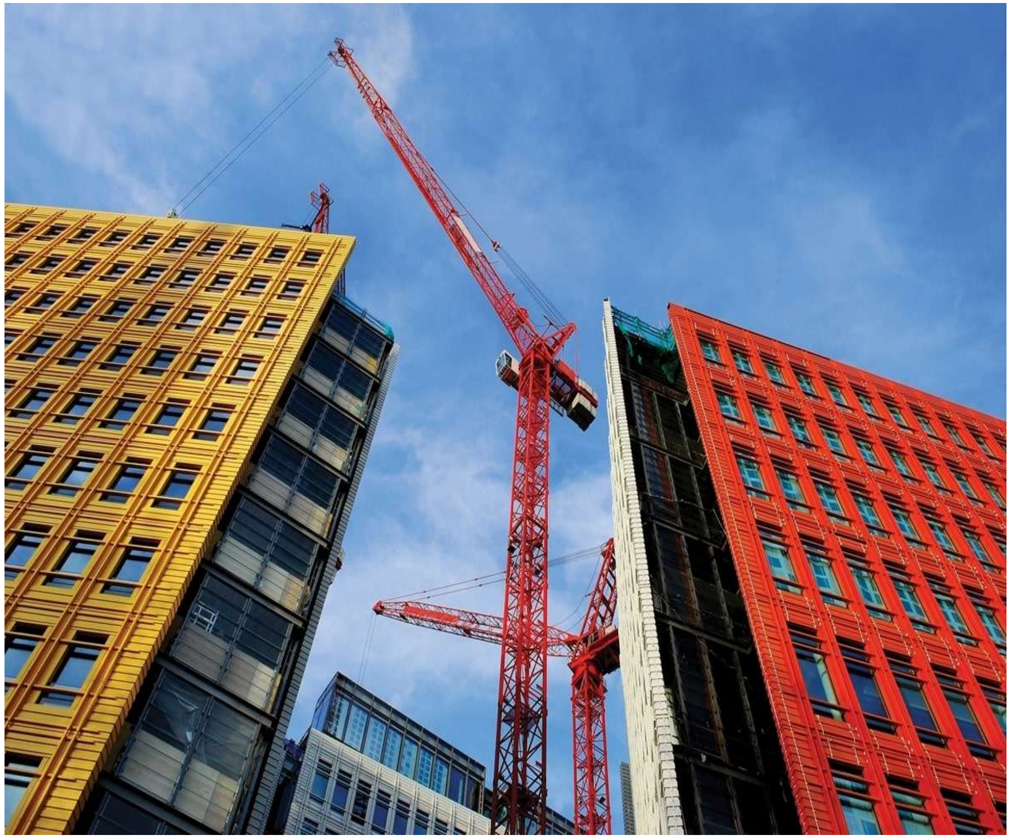
Component description – Tesla T4 Hardware

The Tesla T4 is a professional graphics card by NVIDIA. Built on the 12 nm process, and based on the TU104 graphics processor, the card supports DirectX 12 Ultimate.

- It features 2560 shading units, 160 texture mapping units, and 64 ROPs. Also included are 320 tensor cores which help improve the speed of machine learning applications.

- NVIDIA has paired 16 GB GDDR6 memory with the Tesla T4, which are connected using a 256-bit memory interface. The GPU is operating at a frequency of 585 MHz, which can be boosted up to 1590 MHz, memory is running at 1250 MHz (10 Gbps effective).

- It does not require any additional power connector, its power draw is rated at 70 W maximum.



Building the infrastructure

WSL2 / Ubuntu distribution

1. Ensure that your WSL version is 0.67.6 or newer.
   Systemd support is required!
   To check, run ***wsl --version***.
   To update, run ***wsl --update*** or download from MS Store
2. ***wsl --install***
3. reboot Windows
4. ***wsl --install Ubuntu***
5. wsl --list --verbose

	NAME	STATE	VERSION
*	Ubuntu	Running	2

5. ***wsl***
6. ***sudo apt update; sudo apt upgrade***
7. ***sudo apt install wget tar net-tools mc -y***

Apache Spark (1)

1. Install Java runtime
   Apache Spark requires Java to run
   sudo apt install curl mlocate default-jdk -y

2. Download Apache Spark
Download the latest release of Apache Spark from the downloads page.
https://spark.apache.org/downloads.html

VER=3.5.1 (23. Feb. 2024)
wget https://dlcdn.apache.org/spark/spark-$VER/spark-$VER-bin-hadoop3.tgz
tar xvf spark-$VER-bin-hadoop3.tgz

Move the Spark folder created after extraction to the /opt/ directory.
sudo mv spark-$VER-bin-hadoop3/ /opt/spark

Apache Spark (2)

\# Set Spark environment
\# Open your bashrc configuration file.
nano ~/.bashrc
add:
export SPARK_HOME=/opt/spark
export PATH=$PATH:$SPARK_HOME/bin:$SPARK_HOME/sbin

Activate changes:
source ~/.bashrc

Apache Spark (3)

3. Start a standalone master Server:
start-master.sh
starting org.apache.spark.deploy.master.Master, logging to /opt/spark/logs/spark-root-org.apache.spark.deploy.master.Master-1-EMA.out

The process will be listening on TCP port 8080.
sudo ss -tunelp | grep 8080
tcp LISTEN 0 1 *:8080 *:* users:(("java",pid=5437,fd=286)) ino:61662 sk:6 cgroup:/ v6only:0 <->

http://localhost:8080/

My Spark URL is spark://EMA:7077

Apache Spark (4)

4. Starting Spark Worker Process
The start-worker.sh command is used to start Spark Worker Process.

start-worker.sh spark://EMA:7077

Jupyterlab（1）

pre-requisites

sudo apt install python3 python3-pip python3-venv nodejs -y
python3 --version
Python 3.10.12

pip3 --version
pip 22.0.2 from /usr/lib/python3/dist-packages/pip (python 3.10)

Jupyterlab (2)

<u>add user and group</u>

run the following commands to create a new user called jupyteruser and grant sudo permission

Add a new group
sudo groupadd jupyter
Creating jupyteruser and adding to the jupyter group
sudo useradd --groups jupyter jupyteruser
sudo passwd jupyteruser

add jupyteruser to the sudo group
sudo adduser jupyteruser sudo

sudo chown jupyteruser:jupyter /home/jupyteruser
sudo mkdir /home/jupyteruser
su - jupyteruser

Jupyterlab (3)

```
python3 -m pip install --user --upgrade pip
python3 -m pip install --user psycopg2-binary bokeh plotly chart_studio numpy scipy
python-dotenv
python3 -m pip install --user jupyterlab
python3 -m pip install --user pyspark
python3 -m pip install --user matplotlib seaborn

# install scala kernel

pip install spylon-kernel
sudo python3 -m spylon-kernel install
```

Jupyterlab (4)

https (tls) setup

mkdir ~/ssl_cert && cd ~/ssl_cert
Generate a new private key.
openssl genrsa -out jupyter.key 2048
Create a signed certificate.
openssl req -new -key jupyter.key -out jupyter.csr
Create a self-signed certificate
openssl x509 -req -days 365 -in jupyter.csr -signkey jupyter.key -out jupyter.pem
Certificate request self-signature ok
subject=C = CH, ST = Thurgau, L = Ettenhausen, O = MMS IT GmbH

Jupyterlab（5）

Password protect your JupyterLab server by generating and modifying a Jupyter config file:

jupyter server --generate-config
Writing default config to: /home/jupyteruser/.jupyter/jupyter_server_config.py
jupyter server Password
[JupyterPasswordApp] Wrote hashed password to
/home/jupyteruser/.jupyter/jupyter_server_config.json

Find the config file open it because there are changes required for SSL
nano ~/.jupyter/jupyter_server_config.py
If using the SSL certificate, also add the location of the certificate file and the private key to the config file.

c.ServerApp.certfile = '/home/jupyteruser/ssl_cert/jupyter.pem'
c.ServerApp.keyfile = '/home/jupyteruser/ssl_cert/jupyter.key'

mkdir /home/jupyteruser/notebooks
jupyter-lab --no-browser --ip "*" --notebook-dir=/home/jupyteruser/notebooks --port=8888

Jupyterlab (6)

<u>systemd Setup</u>

sudo nano /etc/systemd/system/jupyter.service
add the following lines:

```
[Unit]
Description=Jupyter Notebook

[Service]
Type=simple
PIDFile=/run/jupyter.pid
# If you need environment variables for Tensorflow GPU work, .bashrc usually does the job
# you need to somehow make those available to the Jupyter service, or else Notebooks that need the GPU won't be able to see it.
Environment="PATH=/usr/local/cuda-12.3/bin:$PATH"
Environment="LD_LIBRARY_PATH=/usr/local/cuda-12.3/lib64:/usr/local/cuda-12.3/lib64:usr/local/cuda/lib64:/usr/local/cuda/extras/CUPTI/lib64:$LD_LIBRARY_PATH"
Environment="CUDA_HOME=/usr/local/cuda-12.3"
```

Jupyterlab (7)

```
Environment="PYSPARK_ALLOW_INSECURE_GATEWAY=1"

Environment="CLASSPATH=/home/jupyteruser/postgresql-42.5.0.jar:$CLASSPATH"

ExecStart=/home/jupyteruser/.local/bin/jupyter-lab --notebook-dir=/home/jupyteruser/notebooks --
no-browser --ip "*" --port=8888
User=jupyteruser
Group=jupyter
Restart=always
RestartSec=10

[Install]
WantedBy=multi-user.target
```

Jupyterlab (8)

sudo systemctl enable jupyter
Created symlink /etc/systemd/system/multi-user.target.wants/jupyter.service → /etc/systemd/system/jupyter.service.

Reload the systemd daemon and restart the service

sudo systemctl daemon-reload
sudo systemctl restart jupyter
sudo systemctl status jupyter

jupyter.service - Jupyter Notebook
     Loaded: loaded (/etc/systemd/system/jupyter.service; enabled; vendor preset: enabled)
     Active: active (running) since Sun 2024-04-07 14:03:11 CEST; 27ms ago
   Main PID: 7507 (jupyter-lab)
      Tasks: 1 (limit: 4589)
     Memory: 2.8M
     CGroup: /system.slice/jupyter.service
             └─7507 /usr/bin/python3 /home/jupyteruser/.local/bin/jupyter-lab --notebook-dir=/home/jupyteruser/notebook>

Apr 07 14:03:11 EMA systemd[1]: Started Jupyter Notebook.

Jupyterlab (9)

Finally, you can monitor the output of the service:
To show the log messages since the last boot (-b) and without additional fields like timestamp and hostname (-o cat), type:

sudo journalctl -u jupyter -b -o cat -f

Open a browser window on your local computer and enter the following to open the notebook.

https://[External IP]:8888

PostgreSQL (1)

apt install postgresql libpostgresql-jdbc-java
systemctl start postgresql
systemctl enable postgresql
systemctl Status PostgreSQL

You will need a JDBC connection to connect Apache Spark to your PostgreSQL database. It’s available for download here:
cd /home/jupyteruser
wget https://jdbc.postgresql.org/download/postgresql-42.7.3.jar
chown jupyteruser:jupyter postgresql-42.7.3.jar

Nvidia CUDA (1)

disable "nouveau" driver because it tries to activate the Tesla card as a graphics card which doesn’t work because it has no graphics port.

In /etc/default/grub, add the following phrase to the value of
GRUB_CMDLINE_LINUX:
module_blacklist=nouveau

Create /etc/modprobe.d/nouveau.conf and add the following line:
blacklist nouveau

Rebuild modules:
depmod -a

Rebuild your grub config:
grub2-mkconfig --output=/boot/efi/EFI/rocky/grub.cfg

Nvidia CUDA (2)

Download and install the Nvidia Tesla driver

wget https://us.download.nvidia.com/tesla/525.60.13/NVIDIA-Linux-x86_64-525.60.13.run
chmod +x *.run
Execute the downloaded package in the Shell
./NVIDIA-xxx --kernel-source-path=/usr/src/kernels/xxx

Nvidia CUDA (3)

nvidia-smi

```
Sat Dec 17 14:03:36 2022
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 525.60.13    Driver Version: 525.60.13    CUDA Version: 12.0     |
|-------------------------------+----------------------+----------------------+
| GPU  Name        Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M. |
|                               |                      |               MIG M. |
|===============================+======================+======================|
|   0  Tesla T4            Off  | 00000000:01:00.0 Off |                    0 |
| N/A   93C    P0    41W /  70W |      2MiB / 15360MiB |      8%      Default |
|                               |                      |                  N/A |
+-------------------------------+----------------------+----------------------+

+-----------------------------------------------------------------------------+
| Processes:                                                                  |
|  GPU   GI   CI        PID   Type   Process name                  GPU Memory |
|        ID   ID                                                   Usage      |
|=============================================================================|
|  No running processes found                                                 |
+-----------------------------------------------------------------------------+
```

Nvidia CUDA (4) – CUDA Toolkit

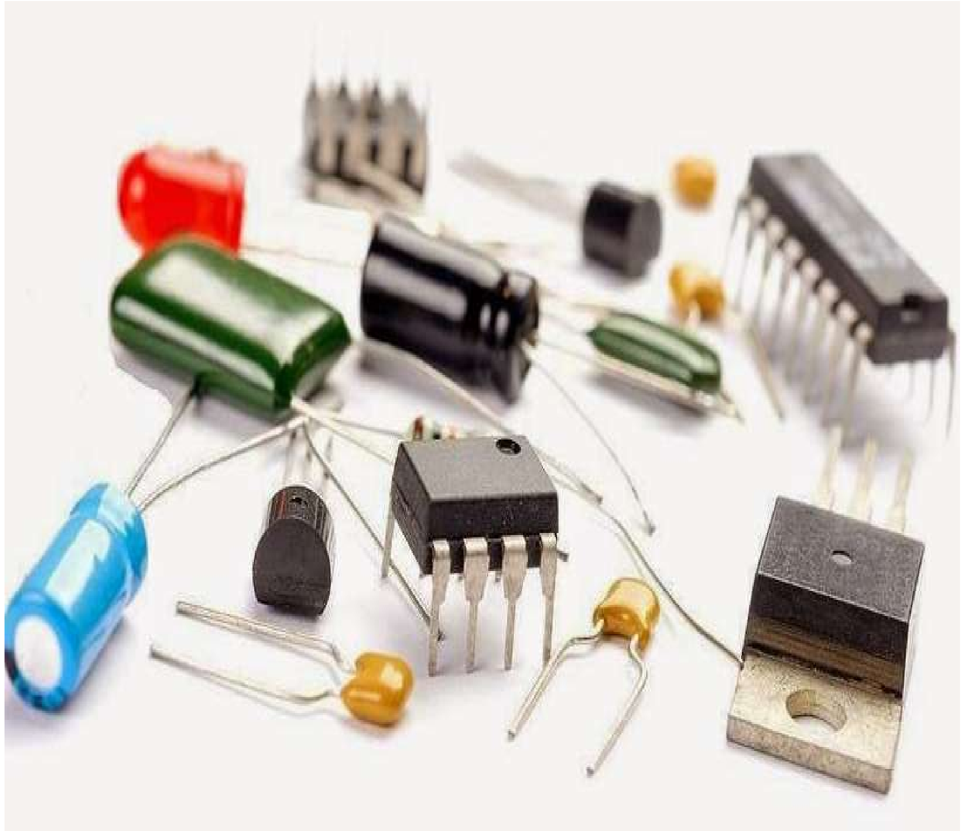
wget https://developer.download.nvidia.com/compute/cuda/10.1/Prod/local_installers/cuda_10.1.243_418.87.00_linux.run
sh cuda_10.1.243_418.87.00_linux.run --override (--override required to bypass gcc version check)
unselect the driver. install the rest

```
===========
= Summary =
===========

Driver:   Not Selected
Toolkit:  Installed in /usr/local/cuda-10.1/
Samples:  Installed in /root/, but missing recommended libraries

Please make sure that
 -   PATH includes /usr/local/cuda-10.1/bin
 -   LD_LIBRARY_PATH includes /usr/local/cuda-10.1/lib64, or, add /usr/local/cuda-10.1/lib64 to
/etc/ld.so.conf and run ldconfig as root

To uninstall the CUDA Toolkit, run cuda-uninstaller in /usr/local/cuda-10.1/bin
```



Demo

Install the demo assets

Download
SMF110_Spark_Python3.ipynb
SMF110_data.json.zip

From

https://github.com/IzODA/examples/tree/master/SMF

and put them into /home/jupyteruser/Notebooks

Demo（1）

2) SMF Data Extraction using Pandas

[4]:

```
#Provide the path of the smf110.json location.
SMF110_PATH = "SMF110_data.json"
smf110_df = pd.read_json(SMF110_PATH)
smf110_df['SMFMNRST'] = pd.to_datetime(smf110_df['SMFMNRST'],unit='ms')
```

Demo（2）

Data Cleaning

[5]:
```
#Convert datatypes.
smf110_df['TRANNUM'] = smf110_df['TRANNUM'].astype(int)
smf110_df['USRCPUT_TIMER'] = smf110_df['USRCPUT_TIMER'].astype(float)
smf110_df['USRCPUT_COUNT'] = smf110_df['USRCPUT_COUNT'].astype(int)

orig_smf110_df = smf110_df.copy(deep=True)

#only keep the CICS regions i.e. SMFMNSPN starting with "CICS".
smf110_df = smf110_df[smf110_df.SMFMNSPN.str.contains("CICS") == True]

smf110_df.head(10)
```

[5]:

	SMFMNSPN	TRAN	TRANNUM	SMF_SID	USRCPUT_TIMER	USRCPUT_FLAG	USRCPUT_COUNT	SMFMNRSD	SMFMI
0	CICS3AAB	Y295	14413	JA0	0.000791	0	29	117302	2017- 09:58:4
1	CICS3AAB	Y295	14414	JA0	0.000729	0	29	117302	2017- 09:58:4

Demo (3)

Exploratory Analysis ¶

[6]:
```
print("The CICS Regions within dataset: ")
orig_smf110_df.SMFMNSPN.unique()
```

```
The CICS Regions within dataset:
```

[6]:
```
array(['CICS3AAB', 'CICS2AAA', 'CICS2AAC', 'CICS5TJA', 'CICS3TAA',
       'CICS2TAB', 'CICS2TAC', 'CICS2AAB', 'CICS1AAB', 'CICS6AAA',
       'CICS2TAA', 'CICS3TAB', 'CICS1TAA', 'CICS6TAA', 'CICS3AAC',
       'CICS6TAB', 'CICS1AAC', 'CICS6AAC', 'CICS5AAC', 'CICS5AJB',
       'CICS6AAB', 'CICS5AAA', 'CICS1AAA', 'CICS5TAA', 'CMASJA0',
       'CICS3AAA'], dtype=object)
```

Demo (4)

```
[7]: print("The number of user tasks in this dataset: " + str(len(smf110_df)))
     print('Total CPU Time in Seconds: {:2f}'.format(smf110_df['USRCPUT_TIMER'].sum()))
     print('Total CPU Time in Hours: {:2f}'.format(smf110_df['USRCPUT_TIMER'].sum() / 3600))
     print('Total DB2 Requests: {}'.format(smf110_df['DB2REQCT'].sum()))
     print('Total WMQ Requests: {}'.format(smf110_df['WMQREQCT'].sum()))
```

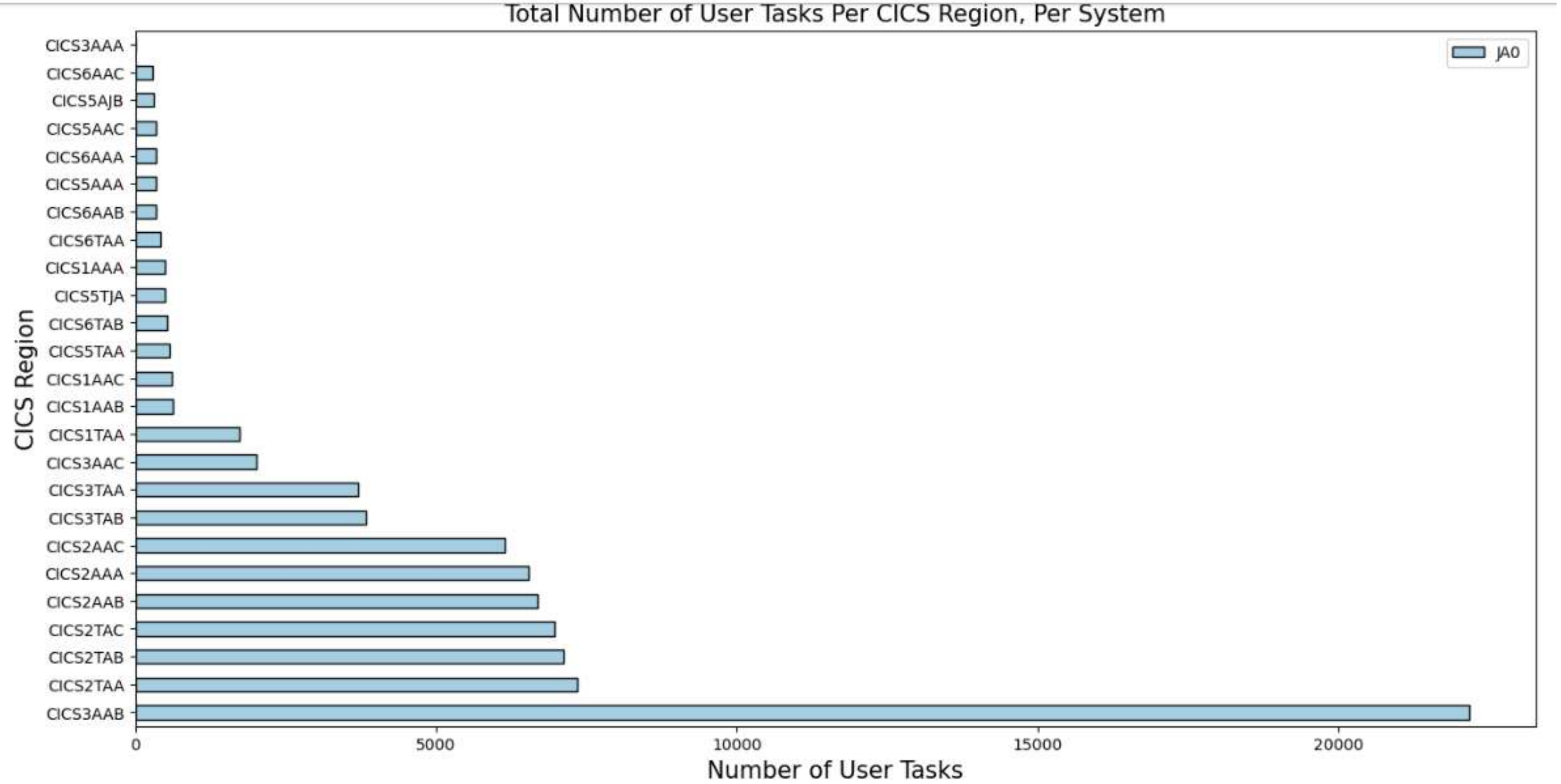
```
The number of user tasks in this dataset: 79955
Total CPU Time in Seconds: 76.903763
Total CPU Time in Hours: 0.021362
Total DB2 Requests: 1111721
Total WMQ Requests: 73959
```

Demo (5)

Total Number of User Tasks Per CICS Region and System

[10]:
```
cics_run_per_region = smf110_df['SMFMNSPN'].value_counts()
plt.figure(figsize=(16,8))
cics_run_per_region.plot.barh(colormap='Paired')
plt.legend(smf110_df['SMF_SID'])
plt.xlabel("Number of User Tasks", fontsize=15)
plt.ylabel("CICS Region", fontsize=15)
plt.title("Total Number of User Tasks Per CICS Region, Per System", fontsize=15)
plt.show()
```

Demo (6)


Total Number of User Tasks Per CICS Region, Per System
JA0
CICS Region
CICS3AAA
CICS6AAC
CICS5AJB
CICS5AAC
CICS6AAA
CICS5AAA
CICS6AAB
CICS6TAA
CICS1AAA
CICS5TJA
CICS6TAB
CICS5TAA
CICS1AAC
CICS1AAB
CICS1TAA
CICS3AAC
CICS3TAA
CICS3TAB
CICS2AAC
CICS2AAA
CICS2AAB
CICS2TAC
CICS2TAB
CICS2TAA
CICS3AAB
0
5000
10000
15000
20000
Number of User Tasks

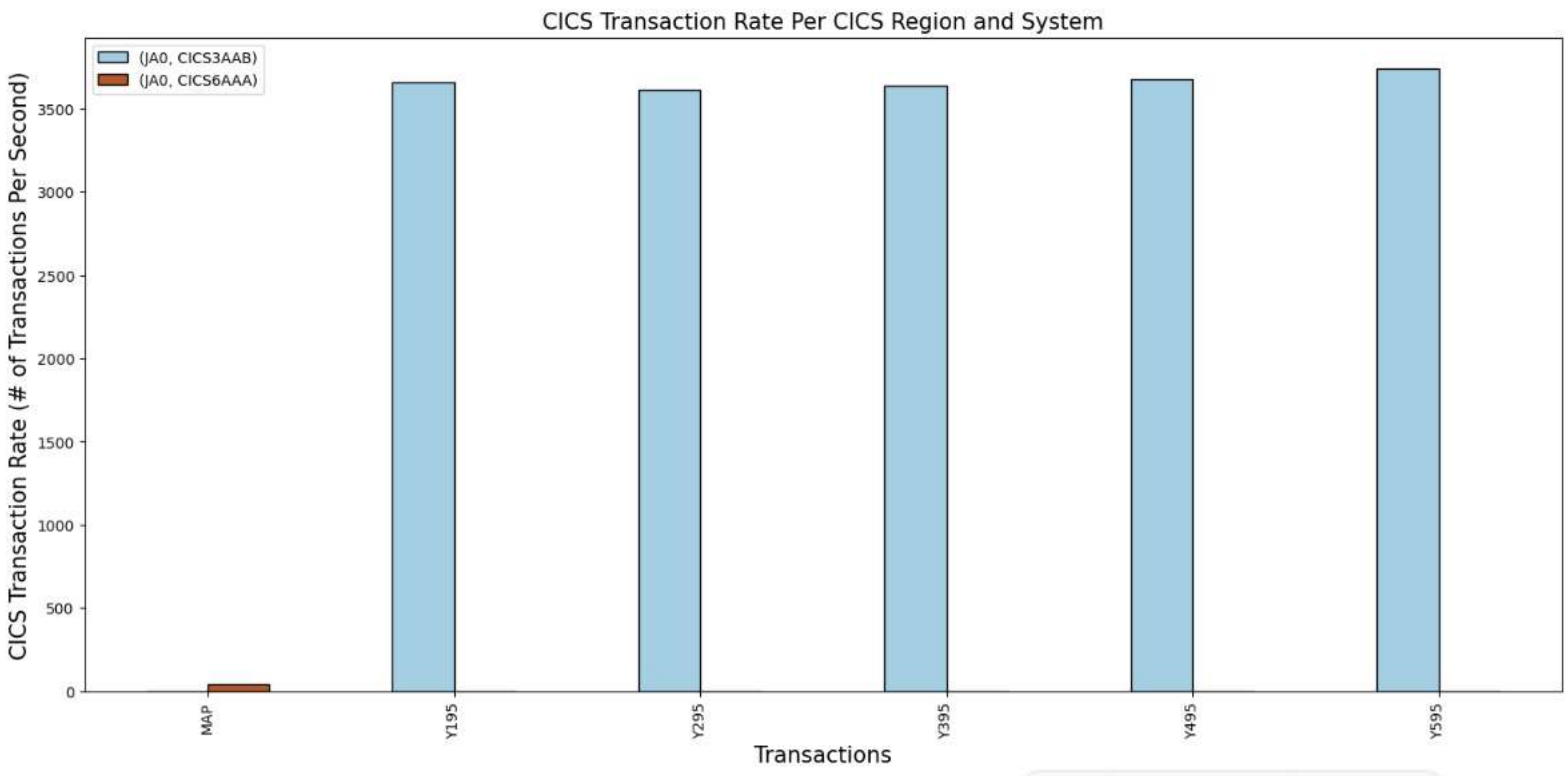
Demo（7）

CICS Transaction Rate Per CICS Region and LPAR

[18]:

```
smf110_filtered_tran_df['DATETIME_SECOND'] = smf110_filtered_tran_df['SMFMNRST'].apply(lambda x: x.second)
trans_rate_per_region = smf110_filtered_tran_df[['SMF_SID','SMFMNSPN','DATETIME_SECOND', 'TRAN']].pivot_table
trans_rate_per_region = trans_rate_per_region.fillna(0)
sys_id = trans_rate_per_region.index.levels[0]
cics_regions = trans_rate_per_region.index.get_level_values(1)
num_days_cics_regions = dict()
for j in range(len(sys_id)):
    for i in range(len(cics_regions)):
        num_days_cics_regions[(sys_id[j],cics_regions[i])] = num_days_cics_regions.get((sys_id[j],cics_region
start_ind = 0
cics_trans_df = pd.DataFrame(index= list(trans_rate_per_region.columns), columns=list(num_days_cics_regions.k
for cics_region_per_sys in num_days_cics_regions:
    end_ind = start_ind + num_days_cics_regions[cics_region_per_sys]
    cics_rate = trans_rate_per_region.iloc[start_ind:end_ind,:].apply(lambda x : np.mean(x))
    cics_trans_df[cics_region_per_sys] = cics_rate
    start_ind = end_ind
cics_trans_df.plot(kind='bar', figsize=(18,8), colormap='Paired')
plt.xlabel("Transactions", fontsize=15)
plt.ylabel("CICS Transaction Rate (# of Transactions Per Second)", fontsize=15)
plt.title("CICS Transaction Rate Per CICS Region and System", fontsize=15)
cics_trans_df
```

Demo（8）



CICS Transaction Rate Per CICS Region and System
(JA0, CICS3AAB)
(JA0, CICS6AAA)
CICS Transaction Rate (# of Transactions Per Second)
3500
3000
2500
2000
1500
1000
500
0
MAP
Y195
Y295
Y395
Y495
Y595
Transactions

10.04.2024