Power Systems VUG Webinar Session 66:

Graphing Temperature, Electricity Use & SSP I/O Stats Data

Nigel Griffiths
Power Systems
Advanced Technology Support
IBM Europe

Host: Jyoti Dodhia
Power Systems
Client Technical Specialist
IBM UK

IBM

Agenda

PART 1
Energy
- What do we want?
- How do we run it?
- What is the output?

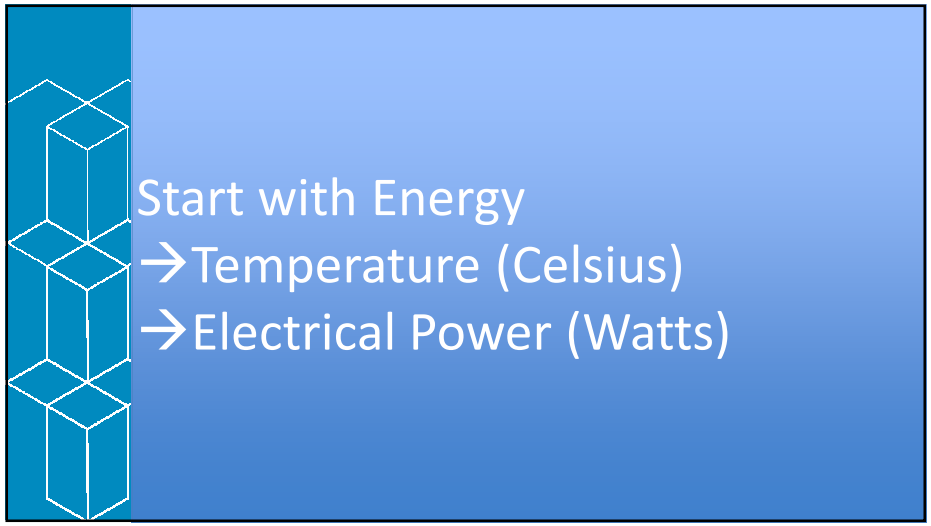
Shared Storage Pools I/O stats
- What do we want?
- How do we run it?
- What is the output?

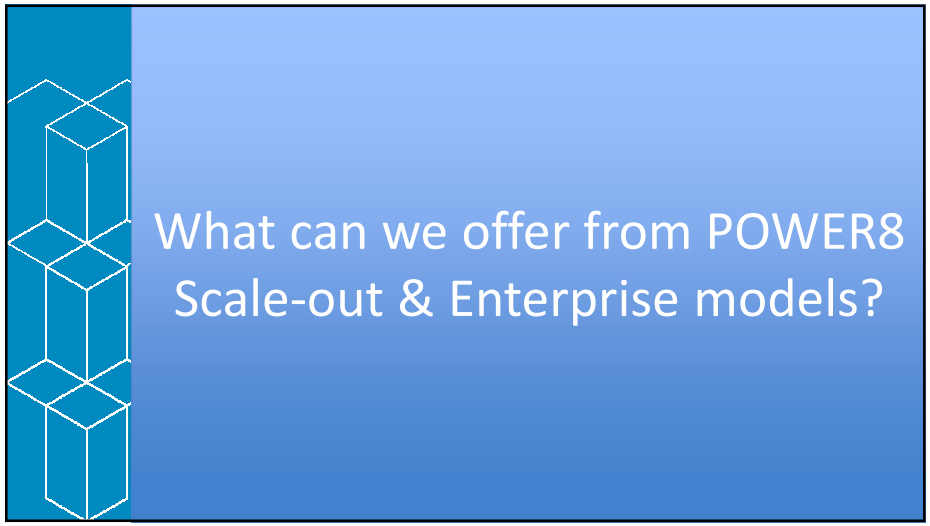
PART 2
Energy Details
- HMC REST API
- Pre-Reqs
- Protocol and data
- Python hints

SSP I/O Details
- Pre-Reqs
- Protocol and data

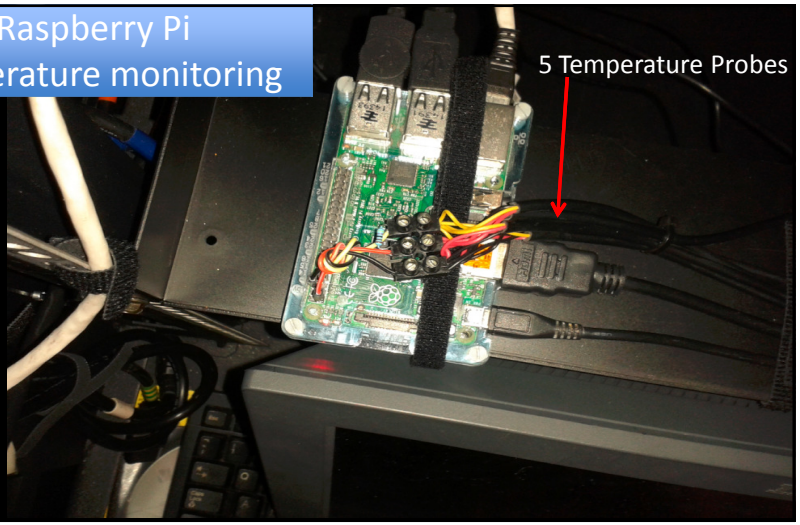
PART 3
REST API basics
Data format
Python Code by a series of samples

Bugs
What is next?
Join the program


Start with Energy
→Temperature (Celsius)
→Electrical Power (Watts)

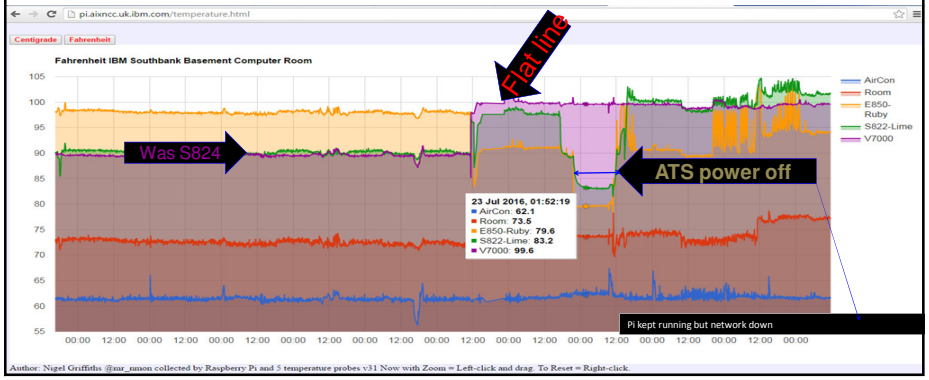

What can we offer from POWER8
Scale-out & Enterprise models?

Raspberry Pi temperature monitoring



What sort of Temperatures?

- Graphed by Googlechart JavaScript library via a automatically built webpage
- Find Details on my AIXpert Blog [Computer Room Temperature Monitoring with a Raspberry Pi]
- 1st August 2016



What do we want?

Per server simple Comma Separated Value (CSV) file
- Electrical consumption in Watts - single number
- Temperature in Celsius - inlet + planar + CPUs

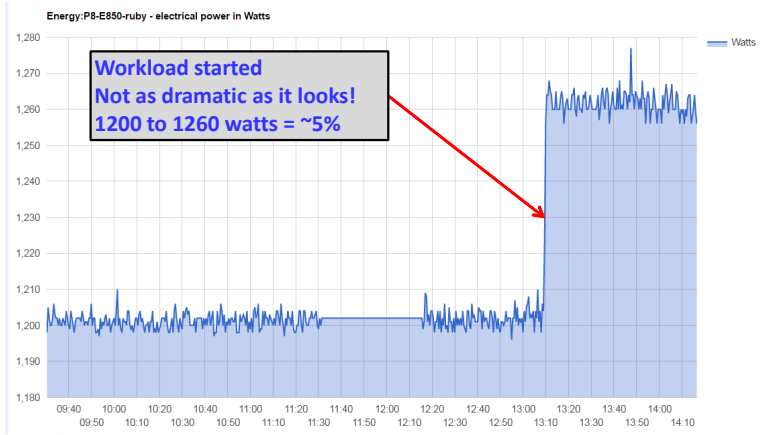
Same data but nicely graphed
- Over time

Don't want to deal with
- Programmers REST API GET/PUT/POST to/from the HMC
- Complicated: XML and JSON file formats
- Writing Python or Java code
- HMC Options: Raw, Aggregated or Processed data types

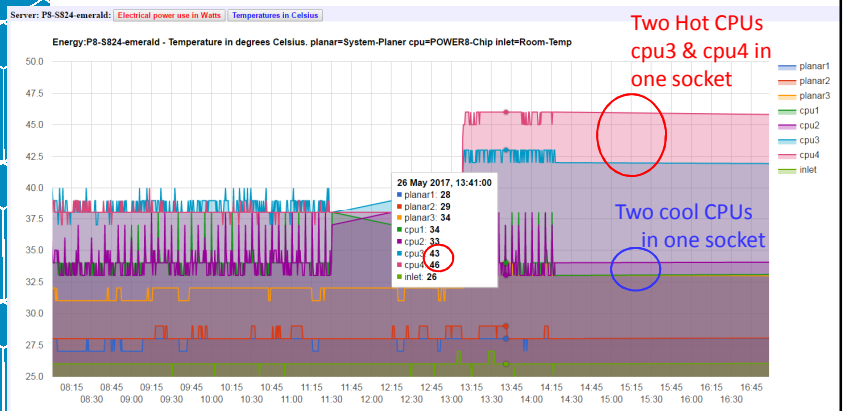
DEMO

http://sky.aixncc.uk.ibm.com/ + Select emerald

Electrical Power Example



Temperature Example



Two Python programs for Energy

Running Python 3.5 on Ubuntu 16.04
- on P8 Supermicro Briggs with 3.5 GHz x 20 CPUs & 256 GB RAM
- Also runs fine on my Raspberry Pi running Ubuntu !
- You can also get Python for AIX from Perzl.org

Scheduled collector:
- energy.py
- Watts and Celsius stats in CSV format per server

Then once generate the graphs
- egoo.py
- Uses the CSV file to generate the webpage per Server

Two Python programs for Energy

1 Collect the stats for one server ~6 seconds

```
./energy.py --hmc hmc14 \
  --username pcmadmin --password SECRET --reuse \
  --server P8-S822-lime          ← HMC name for Managed System
```

2 Generate the .html graphs ~0.2 seconds

```
./egoo.py
```

3 Place .html file on a website

Run energy by cron every hour – for each Server

Then egoo once for the graphs

```
30 * * * * cd /home/nag/Energy ; ./energy.py --hmc hmc14 --username pcmadmin --password  X --reuse --server P8-S822-lime        >> cron.log
31 * * * * cd /home/nag/Energy ; ./energy.py --hmc hmc14 --username pcmadmin --password  X --reuse --server P8-S824-emerald >> cron.log
32 * * * * cd /home/nag/Energy ; ./energy.py --hmc hmc14 --username pcmadmin --password  X --reuse --server P8-E850-ruby    >> cron.log
35 * * * * cd /home/nag/Energy ; ./egoo.py >> cron_egoo.log
```

cd Energy; ./energy.py --hmc hmc14 --username pcmadmin --server P8-S822-lime --reuse --password XXX

Collector energy command syntax

```
./energy.py  --help
usage: energy.py [-h] [--hmc HMC]
   [--username USERNAME]  [--password PASSWORD]
   [--server SERVER] [--debug] [--reuse]
```

Arguments:

--hmc HMC	Specify the HMC IP Address or Hostname
--username USERNAME	Username for the HMC
--password PASSWORD	Password for the HMC
--server SERVER	Server name

Optional:

-h, --help	Show this help message and exit
--debug	Output details as we go + files in directory debug
--reuse	Save and reuse HMC token

Collector energy

It is using the HMC ProcessedMetrics option
~240 data points at 30 second intervals = 2 hours worth

Saved in single file: energy.csv like this sample

P8-S824-emerald,8286-42A,101EC7V,2017-04-31-14-29-30,697,29,29,32,35,34,39,38,28
P8-S824-emerald,8286-42A,101EC7V,2017-04-31-14-30-00,694,29,29,32,34,36,39,38,28
P8-S824-emerald,8286-42A,101EC7V,2017-04-31-14-30-30,710,29,29,32,35,36,40,41,28

Key: Server-name, MTM,Serial-Number, Date time (January=0th month)
then Watts followed by 8 or 13 temperatures

S82* = 8 temps = 3 planar + 4 P8 chips + inlet →2 sockets of P8 2 chips (<24 way)
E850 = 13 temps = 4 planar + 8 P8 chips + inlet →4 sockets of P8 2 chips (<48 way)

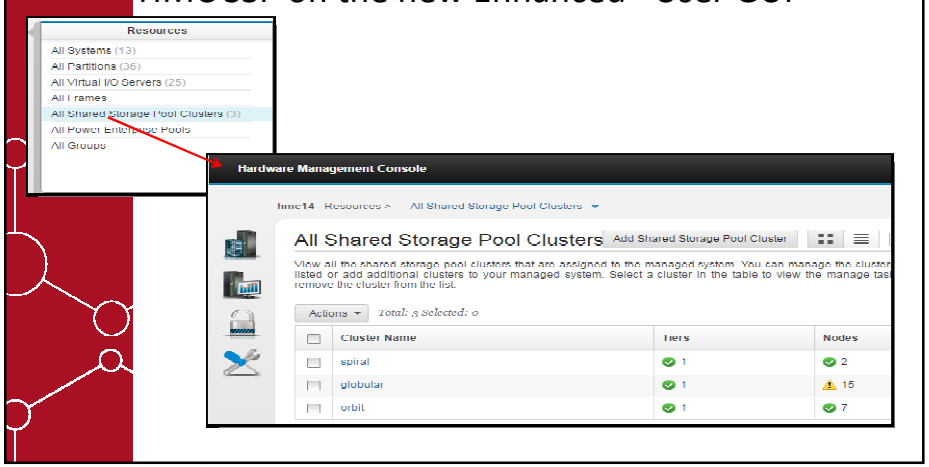
Energy Supported Servers

Supported
- Scale-Out S8nn(L)
- E850

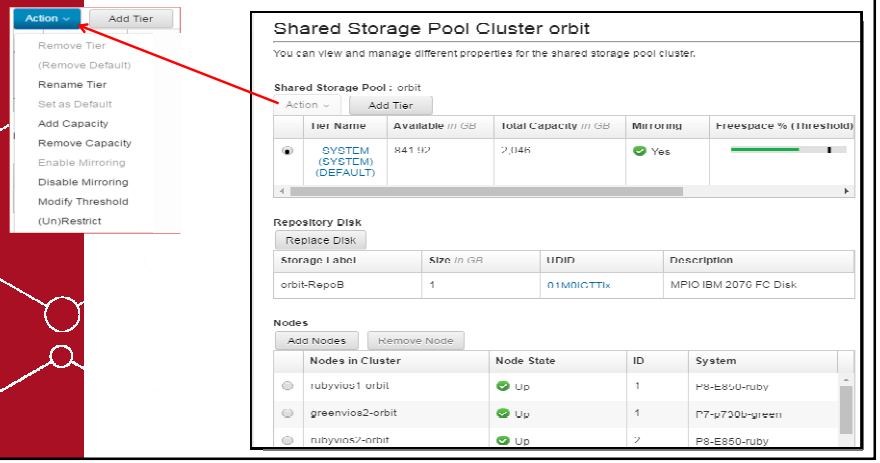
Not supported
- Enterprise E870 / E880

Shared Storage Pools I/O stats

HMC SSP on the new Enhanced+ User GUI



HMC SSP on the new Enhanced+ User GUI

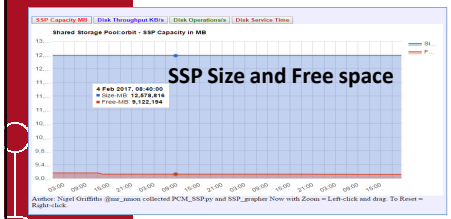


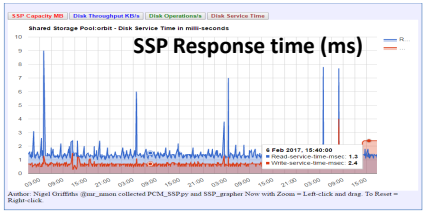
HMC SSP New Function in Dec 2016

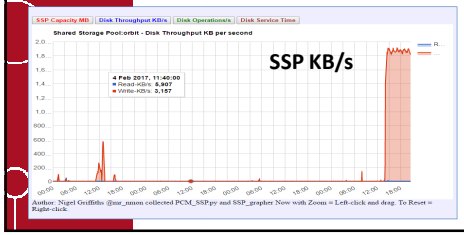
- SSP now supports 24 VIOS nodes
  - –Normal Dual VIOS means 12 Servers
  - –With 24 I/O "busy" VIOS Best Practice include recommendations for the SYSTEM tier to get the best performance
- Question: How to determine if your SSP I/O is lightly used, moderately busy or absolutely hammered?
- Difficult to overview the SSP I/O stats
  - –nmon from 24 VIOSs = hard work to merge!
    - Same LUNs have different hdiskNN on each VIOS
  - –VIOS might be doing non-SSP disk I/O

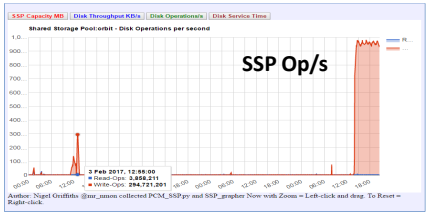
- What you need is Whole SSP I/O + a VIOS break out of those stats.

HMC REST API for Whole SSP level performance stats

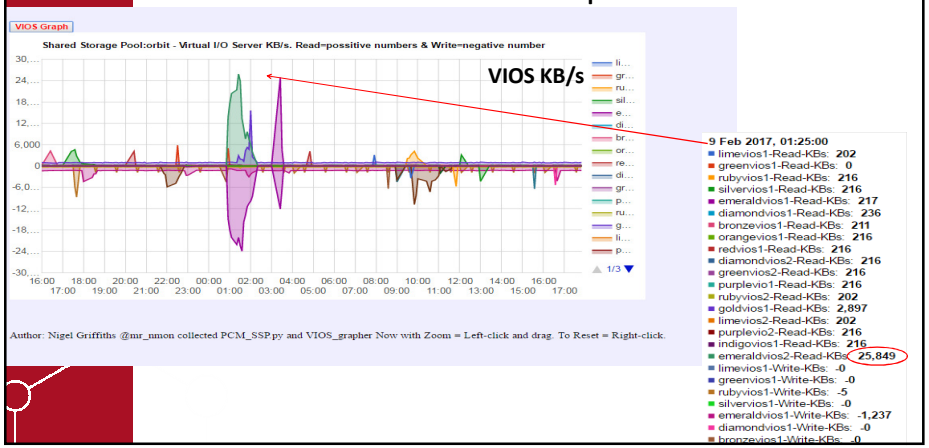








HMC REST API for VIOS level performance stats



Two SSP Python programs

Python 3 like the energy stats

Collect the stats

./SSPIO.py

- does all available SSP on that HMC in one pass

Generate the .html graphs

./goo.py

Run by cron – I suggest every 15 minutes – the data is every 5 minutes

Two Python programs for SSP I/O

1 Collect the stats for one server ~4 seconds

```
./SSPIO.py --hmc hmc14 \
  --username pcmadmin --password SECRET --reuse \
  --prefs                    ← only needs to be included once
```

2 Generate the .html graphs ~0.2 seconds

./goo.py

3 Place .html file on a website

Then goo once for the graphs once

crontab

```
2,17,32,47 * * * * cd /home/nag/SSP ; ./SSPIO4.py --hmc hmc14 --user pcmadmin --password X  >> cron_sspio4.log
4,19,34,49 * * * * cd /home/nag/SSP ; ./goo.py >> cron_goo.log
```

Collector SSP command syntax

$./SSPIO.py --help
usage: SSPIO.py [-h] [--hmc HMC] [--username USERNAME] [--password PASSWORD]
[--ssp SSP] [--debug] [--prefs] [--reuse]

Arguments:

--hmc HMC	Specify the HMC IP Address or Hostname
--username USERNAME	Username for the HMC
--password PASSWORD	Password for the HMC
--ssp SSP	SSP name or ALL(default)
--prefs	Update the SSP preferences (only needed once)

Optional arguments:

--debug	Output details as we go + files in directory debug
--reuse	Save and reuse HMC token
-h, --help	Show this help message and exit

Collector Python program

It is using the ProcessedMetrics option
~290 SSP data points and for my 9 VIOS SSP 1200 VIOS data points
at 5 minute intervals = ~24 hours worth

Saved in SSP total and SSP VIOS files: ssp_total_io.csv like this sample
orbit,2017-04-23-13-10-00,4192256,2304787,2870,2070,1469566,1060003,0.97,1.61
orbit,2017-04-23-13-30-00,4192256,2304787,3000,2739,1536010,1402647,1.01,2.20
orbit,2017-04-23-13-35-00,4192256,2304787,2877,2791,1473204,1429244,1.06,2.14

Key: SSP, Date time (January=0th month) plus Size and free in MB
then read:write KB/s, read:write Op/s, read:write ResponseTime(ms)

VIOS data file ssp_vios_io.csv
Similar but only read:write KB/s for each VIOS
Plus a header line = VIOS names

Part 2:
The Details: Energy stats

How do we get the Stats Overview

REST API for Energy or SSP I/O

- You have to communicate with the HMC as a web browser
  - Open TCPIP socket to communicate
  - GET a file (URL) to request info
  - PUT send information

- Three challenges to tackle in one go
  1. The REST API is complex mostly due to zero worked examples & poor docs
  2. Information is in **XML*** – nests 7 levels deep + namespaces + unexplained
  3. Data in **JSON*** – verbose and vague (missing units) and over large

* Both extremely hard to deal with in ksh script with curl or C

- **Python** has class libraries to help you
- If learning Python that's a 4th challenges ☺

Energy Pre-requisites

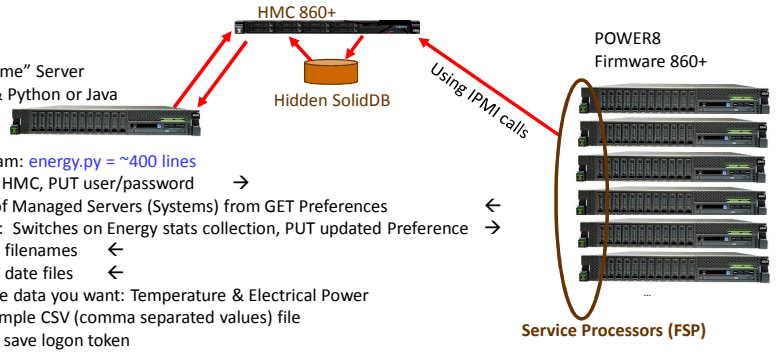
1. HMC with VERY latest 860+ level
2. Network access to HMC (public user GUI network)
3. HMC user id & password with hmcsuperadmin privileges

4. POWER8 HMC attached Scale-out or E850 [Not supported on E870 / E880]
5. POWER8 running 860+ System Firmware

6. Python 3 – probably installed on AIX or Linux or workstation

Recent HMC 860 service packs
- Added in the Preferences XML file: EnergyMonitoringCapable
- This documents the supported servers directly

HMC REST API for Energy stats – pseudo code



Python program: energy.py = ~400 lines
1. Log on to HMC, PUT user/password →
2. Gets list of Managed Servers (Systems) from GET Preferences ←
3. If needed: Switches on Energy stats collection, PUT updated Preference →
4. GET JSON filenames ←
5. GET JSON date files ←
6. Unpick the data you want: Temperature & Electrical Power
7. Save as simple CSV (comma separated values) file
8. Log off or save logon token

Graphing script: egoo.py
1. Sort & remove duplicates in the CSV
2. Reformat into webpage Googlechart .html/Javascript

Data options from HMC

You can request

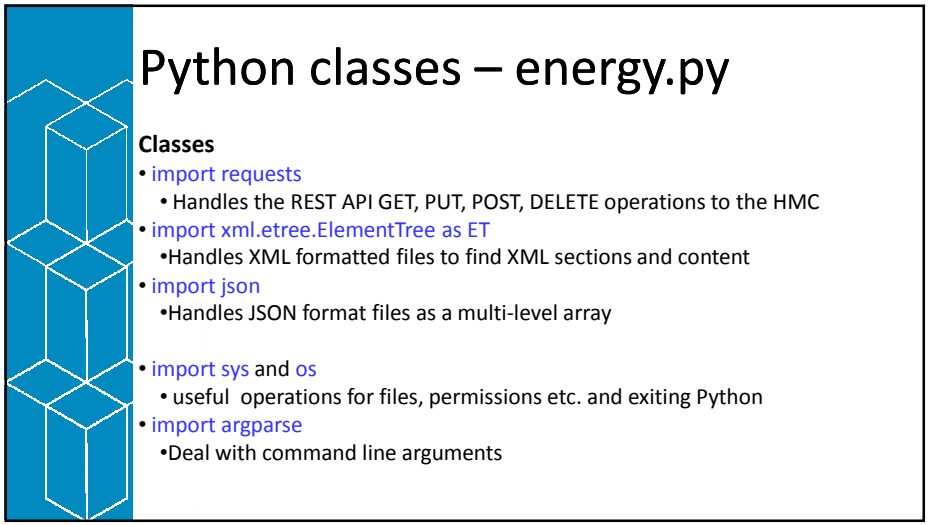
- **Raw data**
  - You have to take differences with the previous set & divide by elapsed time
  - Very hard to code
- **Aggregated data**
  - Gives you calculated stats plus min & max in the period
  - Min + max overkill. If you have 13 stats you now have 39 data points
  - IMHO makes graphs very complex
- **Processed data**
  - Aggregated data but no min +max
  - **This is what I recommend = sane**
  - You get the last 200 – 300 data points
  - So if you miss a data collection or dozens then you get them next time

Data options from HMC

You can request a time date range
– out of the limited data it has (last couple of hours)

- Time date format a pain to calculate
  - Say you want the last 2 hours worth easy right now - 2 hours
    - What it that falls in to yesterday, or last month or Leap year!
  - I still don't know if the date/time is from FSP, HMC or local machine! Wait for the new documentation

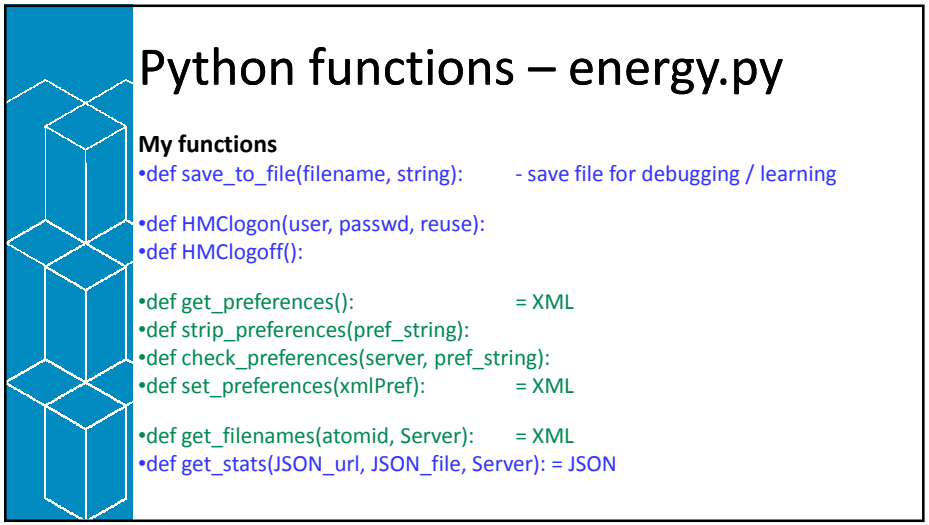
- MUCH easier to get everything and then remove the duplicates later
  - It will return identical data for a particular data point



Python classes – energy.py

Classes

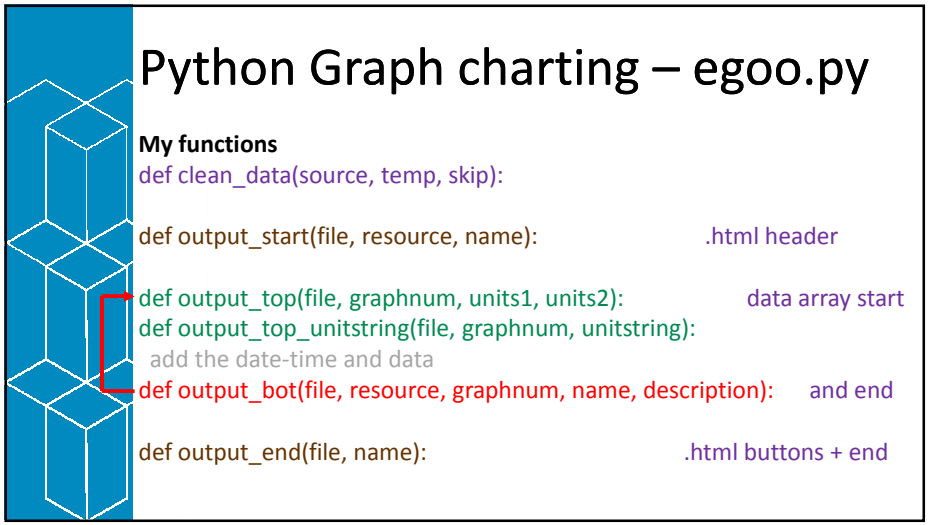
- import requests
  - Handles the REST API GET, PUT, POST, DELETE operations to the HMC
- import xml.etree.ElementTree as ET
  - Handles XML formatted files to find XML sections and content
- import json
  - Handles JSON format files as a multi-level array
- import sys and os
  - useful operations for files, permissions etc. and exiting Python
- import argparse
  - Deal with command line arguments



Python functions – energy.py

My functions

- def save_to_file(filename, string): - save file for debugging / learning
- def HMClogon(user, passwd, reuse):
- def HMClogoff():
- def get_preferences(): = XML
- def strip_preferences(pref_string):
- def check_preferences(server, pref_string):
- def set_preferences(xmlPref): = XML
- def get_filenames(atomid, Server): = XML
- def get_stats(JSON_url, JSON_file, Server): = JSON



Part 2:

The Details:
Shared Storage Pools I/O stats

How do we get the Stats Overview

Same as for Energy Stats

REST API for Energy or SSP I/O

- You have to communicate with the HMC as a web browser
  - Open TCPIP socket to communicate
  - GET a file (URL) to request info
  - PUT send information

- Three challenges to tackle in one go
  1. The REST API is complex mostly due to zero worked examples & poor docs
  2. Information is in **XML*** – nests 7 levels deep + namespaces + unexplained
  3. Data in **JSON*** – verbose and vague (missing units) and over large

* Both extremely hard to deal with in ksh script with curl or C

- **Python** has class libraries to help you
- If learning Python that's a 4th challenge ☺

SSP Pre-requisites

Starter pack

- HMC with VERY latest 860+ level
- **Shared Storage Pool based on VIOS 2.2.5.20+**
-
- Network access to HMC (public network)
- HMC user id & password with hmcsuperadmin privileges

- Access to Python 3– probably installed on AIX or Linux
  - If coding: Good **Python** books or online manuals/examples

Data options from HMC

Same Raw, Aggregated and Processed and data/time range options
same as the Energy stats
Again **Processed stat** are recommended

Classes, Functions and Graphing

All the same as for Energy – see previous charts

The Preferences XML is handled a different way
- Due to the XML format (two levels of metadata)

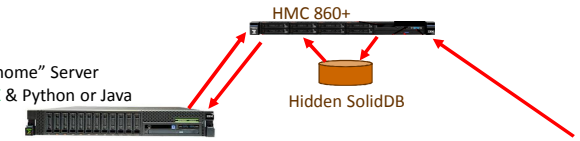
Data options from HMC

SSP only feature: stats detail depth all in one large JSON format file
JSON files – frequency 5 minutes
Whole **SSP level stats** ← **good**
Tier level ← If you have tiers = not implemented in SSPIO.py (yet)
VIOS level ← **good**
Disks level ← Swamped 24 VIOS nodes x 64 LUNs = 1536 disks

SSP Level
- SSP Size & free space
- Read:&Write: KB/s, ops/s, Service-Time
- Skipping lots of diagnostic level stats: timeouts, failures ← for a PMR!

VIOS level – only Read:Write for KB/s

HMC REST API for SSP I/O stats – pseudo code



Python program
1. Log on to HMC, PUT user/password →
2. Gets list of SSPs from GET Preferences ←
3. If needed: Switches on SSP stats collection, PUT updated Preference →
4. GET JSON filenames ←
5. GET JSON date files ←
6. Unpick the data you want: Whole SSP and VIOS levels (ignore disk level)
7. Save as simple CSV (comma separated values)
8. Log off or save token

Graphing script
1. Sort & remove duplicates in the CSV
2. Reformat into webpage Googlechart .html/Javascript

VIOS in the SSP Cluster ALL running 2.2.5.20 +
Note: 1 VIOS gathers the stats from the whole SSP & send it to the HMC

Part 3:

The Deep Dive:

BUGs

REST API basics

File formats

Python code

Very simple Request to a website

Very simple to a website using Python "requests" loadable module

```
>>> import requests
>>> ret = requests.get('http://google.com/index.html')
>>> print(ret.status_code)
200
>>> print(ret.text)
```

Very simple Request to a website

```
# python3
>>> import requests                                  ← load library
>>> ret = requests.get('http://google.com/index.html')  ← connect to Google
>>> print(ret.status_code)                           & ask for file index.html
200                                                  ← 200 = OK
>>> print(ret.text)
<html><head>
<title>Google</title>                                ← browser Tab name
</head>
<body>
<h1>Google</h1>                                      ← large print heading
<img src=http://www.google.co.uk/images/logo.gif>    ← image to fetch
<a href="http://www.google.co.uk/img.html>Images</a> ← Link
<script> JAVA-SCRIPT HERE </script>                  ← used for search
</body></html>                                         box on screen
```

Very simple Request to a website

Very simple to a website using Python "requests" loadable module

```
>>> import requests
>>> ret = requests.get('http://sky.aixncc.uk.ibm.com/index.html')
>>> print(ret.status_code)
200
>>> print(ret.text)
```

→

```
<HTML>
<TITLE>nweb</TITLE>
<BODY BGCOLOR="lightblue">
<H1>nweb Test page</H1>
<IMG SRC="nigel.jpg">
<p>
Not pretty but it should prove that nweb works :-)
<h1> CURRENT Googlechart graphs of HMC REST API supplied data</H1>
<b>Energy (Watts and Celsius) </b>
<ul>
<li><a href="/energyP8-S824-emerald.html"> P8-S824-emerald </a>
<li><a href="/energyP8-S822-lime.html"> P8-S822-lime </a>
<li><a href="/energyP8-E850-ruby.html"l> P8-E850-ruby </a>
</ul>
<b>Shared Storage Pools (Size/free, KB/s, operations/s,
service-time and VIOS stats) </b>
<ul>
<li><a href="/ssp_orbit.html"> SSP: orbit </a>
<li><a href="/ssp_spiral.html"> SSP: spiral </a>
</ul>
</BODY>
</HTML>
```

HMC login

```
>>> import requests
>>> import xml.etree.ElementTree as ET

>>> header = {'Content-Type': 'application/vnd.ibm.powervm.web+xml;  type=LogonRequest'}
>>> url =  'https://hmc14.ibm.com:12443/rest/api/web/Logon'
>>> payload = '<LogonRequest schemaVersion="V1_0"
        xmlns="http://www.ibm.com/xmlns/systems/power/firmware/web/mc/2012_10/"
    xmlns:mc="http://www.ibm.com/xmlns/systems/power/firmware/web/mc/2012_10/">
    <UserID>nigel</UserID><Password>SECRET</Password></LogonRequest>'

>>> ret = requests.put(url, data=payload, headers=header, verify=False)
>>> print(ret.status_code)
200
>>> xmlResponse = ET.fromstring(ret.text)
>>> token = xmlResponse[1].text
>>> print(token)
```

← convert test string to XML

yU7fZEnsKoW4sJhcWCzUeRwKpZA1VpgBacFYAm5pBTHhmFv3kJ_56AlpQ3MKeP4TMvBkjRIdBvdsgLBKsSjrMwFksQ4WHpYuOhCOUIE0PAk17559F9pgMrvyeKLhCMjlAtffnANtWDLzgxWY6mQeGcFZN9hN75Ph46CDCD9A0MpW6I_CifH57mtitn1JG3_TUymCBWOO2w0fE5Sj39NL4vucCyqT8TP3FRqIUKr_8b4=

HMC logoff

We could save the token in a file and reuse it later, when the script runs again.
- This is a small security risk so keep that file locked down.
- It will remain valid for a few days.

This keeps a HMC session active.
You can end up with 1000's of old sessions!
Or you can disconnect - in REST API terms it is a "delete of the Logon" !

```
>>> header = {'X-API-Session' : token }
>>> url =  'https://hmc14.ibm.com:12443/rest/api/web/Logon'

>>> ret = requests.delete(url, headers=header, verify=False)
>>> print(ret.status_code)
204                    ← 204 = No Content returned = OK    200 also OK
```

HMC get preferences

```
>>> import requests
>>> import xml.etree.ElementTree as ET

>>> header = {'X-API-Session' : token}
>>> url = 'https://hmc14.ibm.com:12443/rest/api/pcm/preferences'

ret = requests.get(url, headers=header, verify=False)
>>> ret = requests.put(url, headers=header, verify=False)
>>> print(ret.status_code)
200
>>> xmlResponse = ET.fromstring(ret.text)      ← convert test string to XML
>>> print(xmldata)
# it can't print as it is not a native Python data structure
```

Get Preferences → HMC XML data

```
<feed xmlns="http://www.w3.org/2005/Atom" xmlns:ns2="http://a9.com/-/spec/opensearch/1.1/"
xmlns:ns3="http://www.w3.org/1999/xhtml">
  STUFF
  <title type="text">Performance and Capacity Monitoring Preferences</title>
  STUFF
<entry>
    STUFF
    <content type="application/xml">
      <ManagementConsolePcmPreference STUFF STUFF STUFF>          ← HMC Level
  STUFF
  <MaximumManagedSystemsForLongTermMonitor kxe="false" kb="ROR">45</MaximumManagedSystemsForLongTermMonitor>
  <MaximumManagedSystemsForComputeLTM kxe="false" kb="ROR">60</MaximumManagedSystemsForComputeLTM>
  <MaximumManagedSystemsForAggregation kxe="false" kb="ROR">40</MaximumManagedSystemsForAggregation>
  <MaximumManagedSystemsForShortTermMonitor kxe="false" kb="ROR">5</MaximumManagedSystemsForShortTermMonitor>
  <MaximumManagedSystemsForEnergyMonitor kb="ROR" kxe="false">45</MaximumManagedSystemsForEnergyMonitor>
  <AggregatedMetricsStorageDuration kb="UOD" kxe="false">31622400</AggregatedMetricsStorageDuration>
<ManagedSystemPcmPreference kb="UOD" kxe="false" schemaVersion="V1_5_1">   ← Server Level
STUFF about this Server
 </ManagedSystemPcmPreference>          REPEATED FOR EACH SERVER ON THIS HMC
 ....
 ....
</ManagementConsolePcmPreference>
    </content>
  </entry>
</feed>
```

Get Preferences → shrunk HMC XML data

```
def strip_preferences(pref_string):
    i = pref_string.find("<ManagementConsolePcmPreference")
    pref_string = pref_string[i :  ]
    i = pref_string.find("</content>")
    pref_string = pref_string[ :i ]
    return pref_string
```

```
<feed xmlns="http://www.w3.org/2005/Atom" xmlns:ns2="ht
xmlns:ns3="http://www.w3.org/1999/xhtml">
  STUFF
  <title type="text">Performance and Capacity Monitoring Pre
  STUFF
<entry>
    STUFF
    <content type="application/xml">
      <ManagementConsolePcmPreference STUFF STUFF STUFF>          ← HMC Level
  STUFF
  <MaximumManagedSystemsForLongTermMonitor kxe="false" kb="ROR">45</MaximumManagedSystemsForLongTermMonitor>
  <MaximumManagedSystemsForComputeLTM kxe="false" kb="ROR">60</MaximumManagedSystemsForComputeLTM>
  <MaximumManagedSystemsForAggregation kxe="false" kb="ROR">40</MaximumManagedSystemsForAggregation>
  <MaximumManagedSystemsForShortTermMonitor kxe="false" kb="ROR">5</MaximumManagedSystemsForShortTermMonitor>
  <MaximumManagedSystemsForEnergyMonitor kb="ROR" kxe="false">45</MaximumManagedSystemsForEnergyMonitor>
  <AggregatedMetricsStorageDuration kb="UOD" kxe="false">31622400</AggregatedMetricsStorageDuration>
<ManagedSystemPcmPreference kb="UOD" kxe="false" schemaVersion="V1_5_1">   ← Server Level
STUFF
 </ManagedSystemPcmPreference>          REPEATED FOR EACH SERVER ON THIS HMC
 ....
 ....
</ManagementConsolePcmPreference>
    </content>
  </entry>
</feed>
```

Get Preferences → HMC XML Server data

```
<ManagedSystemPcmPreference kb="UOD" kxe="false" schemaVersion="V1_5_1">
    <Metadata>
      <Atom>
        <AtomID>caac7a03-4eac-3256-b32b-5b203809d674</AtomID>
        <AtomCreated>1496854761228</AtomCreated>
      </Atom>
    </Metadata>
    <SystemName kb="ROR" kxe="false">P8-E850-ruby</SystemName>          ← Server names as seen on the HMC
    <MachineTypeModelSerialNumber kxe="false" kb="ROR" schemaVersion="V1_5_1">
      <Metadata>
        <Atom/>
      </Metadata>
      <MachineType kb="ROR" kxe="false">8408</MachineType>             ← Machine Type   (MTM)
      <Model kxe="false" kb="ROR">E8E</Model>                          ← Model          (MTM)
      <SerialNumber kxe="false" kb="ROR">21D594V</SerialNumber>        ← SerialNUmber
    </MachineTypeModelSerialNumber>
    <EnergyMonitoringCapable kxe="false" kb="ROO">true</EnergyMonitoringCapable>   ← Read only information
    <LongTermMonitorEnabled kb="UOD" kxe="false">true</LongTermMonitorEnabled>     ← Set true automatically with the below
    <AggregationEnabled kb="UOD" kxe="false">true</AggregationEnabled>             ← Set true automatically with the below
    <ShortTermMonitorEnabled kb="UOD" kxe="false">false</ShortTermMonitorEnabled>
    <ComputeLTMEnabled ksv="V1_1_0" kxe="false" kb="UOD">true</ComputeLTMEnabled>
    <EnergyMonitorEnabled kb="UOD" kxe="false">true</EnergyMonitorEnabled>         ← Is collecting stats (default is false)
    <AssociatedManagedSystem kb="ROO" kxe="false"
href="https://hmc14.aixncc.uk.ibm.com:12443/rest/api/uom/ManagedSystem/caac7a03-4eac-3256-b32b-5b203809d674"
rel="related"/>
  </ManagedSystemPcmPreference>
```

Get Preferences Python access to HMC data

```
<ManagementConsolePcmPreference STUFF STUFF STUFF>          ← HMC Level
  STUFF
<ManagedSystemPcmPreference kb="UOD" kxe="false" schemaVersion="V1_5_1">   ← Server Level
  STUFF
  <SystemName kb="ROR" kxe="false">P8-E850-ruby</SystemName>
  <EnergyMonitoringCapable kxe="false" kb="ROO">true</EnergyMonitoringCapable>
  <EnergyMonitorEnabled kb="UOD" kxe="false">true</EnergyMonitorEnabled>
</ManagedSystemPcmPreference>

<ManagedSystemPcmPreference kb="UOD" kxe="false" schemaVersion="V1_5_1">   ← Server Level
  STUFF
  <SystemName kb="ROR" kxe="false">P8-S822-lime</SystemName>
  <EnergyMonitoringCapable kxe="false" kb="ROO">true</EnergyMonitoringCapable>
  <EnergyMonitorEnabled kb="UOD" kxe="false">false</EnergyMonitorEnabled>
</ManagedSystemPcmPreference>

Repeat …

</ManagementConsolePcmPreference>
```

```
for server in root.findall('ManagedSystemPcmPreference'):
        name      = server.find('SystemName')
        capable   = server.find('EnergyMonitoringCapable')
        enabled   = server.find('EnergyMonitorEnabled')
        print('FOUND System=%s Capable=%s Enabled=%s'
              %(name.text, capable.text, enabled.text))

FOUND System= P8-E850-ruby  Capable= true  Enabled= true
FOUND System= P8-S822-lime  Capable= true  Enabled= false
. . .
Set the flag like this:
        enable.text = `true`
```

Preferences Aliases & Name spaces

Actual code a little more complex due to Aliases (aaa:bbb) and name spaces (ns) plus we need the AtomID later but it is nested 3 levels further in !!

```
ns = {'yyy': 'http://www.ibm.com/xmlns/systems/power/firmware/pcm/mc/2012_10/'}

for this in root.findall('yyy:ManagedSystemPcmPreference',ns):
        name      = this.find('yyy:SystemName',ns)
        aggregate = this.find('yyy:AggregationEnabled',ns)
        capable   = this.find('yyy:EnergyMonitoringCapable',ns)
        enabled   = this.find('yyy:EnergyMonitorEnabled',ns)
        print('FOUND System=%32s Aggregation=%6s Capable=%6s Enabled=%6s'
            %(name.text, aggregate.text, capable.text, enabled.text))

        meta   = this.find('yyy:Metadata',ns)
        atom = meta.find('yyy:Atom',ns)
        atomitem = atom.find('yyy:AtomID',ns)

        if server == name.text:
            result = 'found'
            AtomID = atomitem.text
            if capable.text == 'true':
                print("    Server can do Energy monitoring")
                aggregate.text = 'true'
                enabled.text = 'true'
```

Set Preferences to switch a flag to 'true'

```
>>> import requests
>>> import xml.etree.ElementTree as ET

>>> head = {'Content-Type': 'application/xml', 'X-API-Session' : token }
>>> url = 'https://hmc14.ibm.com:12443/rest/api/pcm/preferences'
```

After setting the flags switch on Energy data

```
>>> xmlStr = ET.tostring(xmlPreference)
>>> ret = requests.post(url, headers=head, data=xmlStr, verify=False)

>>> print(ret.status_code)
200
```

← convert XML to a character string to send back to the HMC

OK that is the easy part done!

Next: using that AtomID we can request the filename(s) of the data

Get filenames

```
>>> import requests
>>> import xml.etree.ElementTree as ET

>>>  head = {'X-API-Session' : token, 'Accept': 'application/atom+xml' }
>>> url =  'https://hmc14.ibm.com:12443/rest/api/pcm/ManagedSystem/'+AtomId+'/ProcessedMetrics?Type=Energy'
```

Example AtomId= caac7a03-4eac-3256-b32b-5b203809d674

```
>>> ret = requests.get(url, headers=head, verify=False)

>>> print(ret.status_code)
200
```

Important:
The HMC prepares the stats data in to file(s) on the HMC this can take a few seconds
Data returned is the filenames and URLs to get the files
The data returned is “of course” XML !!

Get filenames

```
<feed xmlns="http://www.w3.org/2005/Atom" xmlns:ns2="http://a9.com/-/spec/opensearch/1.1/"
xmlns:ns3="http://www.w3.org/1999/xhtml">
  <id>6ed2156b-342c-38cc-93d2-e01dad3c3fc7</id>
  <updated>2017-05-31T19:51:46.138Z</updated>
  <title type="text">ProcessedMetrics</title>
  <subtitle type="text">ManagedSystem 6ed2156b-342c-38cc-93d2-e01dad3c3fc7</subtitle>
  <link rel="self" href="https://hmc14:12443/rest/api/pcm/ManagedSystem/6ed2156b-342c-38cc-93d2-
e01dad3c3fc7/ProcessedMetrics?Type=Energy"/>
  <generator uri="IBM Power Systems Management Console" version="1"/>
  <entry>
    <id>4c5ca506-a841-476f-aff6-dc3ac9c492c0</id>
    <title type="text">EnergyMetrics_ManagedSystem_6ed2156b-342c-38cc-93d2-e01dad3c3fc7_null_null_30.json</title>
    <link type="application/json"
href="https://hmc14:12443/rest/api/pcm/ProcessedMetrics/EnergyMetrics_ManagedSystem_6ed2156b-342c-38cc-93d2-
e01dad3c3fc7_null_null_30.json"/>
    <author>
      <name>IBM Power Systems Management Console</name>
    </author>
    <category term="ManagedSystem" frequency="30"/>
  </entry>
</feed>
```

Used as a filename if we later want to save a JSON file

Used to get the JSON file from the HMC

Warnings:

Note there are multiple <link> and <titles> sections at different levels to make this harder!

Raw stats can return many dozens of filenames in one XML file.

Extract filenames from XML & get JSON

```
>>> …
>>>filenames_xml = ET.fromstring(filenames_str)
>>>for reffeed in filenames_xml:
>>>   if 'entry' in reffeed.tag:
>>>      for entry in reffeed:
>>>         if 'title' in entry.tag:
>>>            filename = entry.text          ← useful if you save the data to a file
>>>         if 'link' in entry.tag:
>>>            JSON_file_url = entry.get('href')   ← what you need to get the data

>>> head = {'X-API-Session' : token }
>>>  ret = requests.get(JSON_file_url, headers=head, verify=False)

>>> print(ret.status_code)
200
```

The data returned is in JSON format = a whole new ball game

We are nearly there!

We have the stats data
- Assuming it did not give us a file with an error message!

Next we go into the JSON rabbit hole:

JSON default is no newline characters = 1 line = impossible to edit
The fix is to reformat the file:
$ python3 -m json.tool <myfile.json >readable.json
Python json class is a nested text-addressable array

JSON format

```
{
  "systemUtil": {
    "utilInfo": {            ← one header
      "frequency": 30,
      "metricArrayOrder": [ "AVG" ], ← Processed only has averages
      "metricType": "Processed",
      "mtms": "8284-22A*215296V",
      "name": "P8-S822-lime",
      "uuid": "6ed2156b-342c-38cc-93d2-e01dad3c3fc7",
      "version": "1.3.0"
    },
    "utilSamples": [   ← long array of samples each with Watts + Celsius
      {
        "energyUtil": {
          "powerUtil": {
            "powerReading": [ 452]          ← Watts - 1 per sample
          },
          "thermalUtil": {
            "baseboardTemperatures": [
              {
                "entityId": "Baseboard temperature sensors(42h)",
                "entityInstance":"1",
                "temperatureReading": [ 27 ]  ← Celsius - many per sample
              },
. . .
. . .
              "cpuTemperatures": [
              {
                "entityId": "CPU temperature sensors(41h)",
                "entityInstance":"4",
                "temperatureReading": [ 36 ]
              },
. . .
. . .
              "inletTemperatures": [
              {
                "entityId": "Inlet air temperature(40h)",
                "entityInstance":"1",
                "temperatureReading": [ 22 ]  ← Celsius
              }
            ]
          }
        },
        "sampleInfo": {← one trailer
          "status": 0,        ← Zero is good
          "timeStamp": "2017-05-25T15:00:30+0100"
        },
        "sampleType": "ManagedSystem"
      }
    ]   ← End of array of samples
  }
}
```

JSON format

```
{
  "systemUtil": {
    "utilInfo": {            ← one header
      "frequency": 30,
      "metricArrayOrder": [ "AVG"], ← Processed only has averages
      "metricType": "Processed",
      "mtms": "8284-22A*215296V",
       "name": "P8-S822-lime",        → systemUtil[utilInfo][name]
      "uuid": "6ed2156b-342c-38cc-93d2-e01dad3c3fc7",
      "version": "1.3.0"
    },
    "utilSamples": [   ← long array of samples each with Watts + Celsius
      {                                  for sample in data[systemUtil][utilSamples]
        "energyUtil": {
          "powerUtil": {
            "powerReading": [ 452]              ← Watts - 1 per sample
          },   →sample[energyUtil][powerUtil][powerReading][0]
          "thermalUtil": {
            "baseboardTemperatures": [
              {
                "entityId": "Baseboard temperature sensors(42h)",
                "entityInstance": "1",
                "temperatureReading": [ 27 ]  ← Celsius - many per sample
              },
→sample[thermalUtil][baseboardTemperatures][temperatureReading][0]
. . .
. . .
            "cpuTemperatures": [
              {
                "entityId": "CPU temperature sensors(41h)",
                "entityInstance": "4",
                "temperatureReading": [ 36 ]
              },
→sample[thermalUtil] \
    [cpuTemperatures][temperatureReading][0]
. . .
            "inletTemperatures": [
              {
                "entityId": "Inlet air temperature(40h)",
                "entityInstance": "1",
                "temperatureReading": [ 22 ]  ← Celsius
              }
→sample[thermalUtil] \
    [inletTemperatures][temperatureReading][0]
            ]
          }
        },
        "sampleInfo": {← one trailer
          "status": 0,        ← Zero is good
          "timeStamp": "2017-05-25T15:00:30+0100"
        }, →sample[sampleInfo][timeStamp]
        "sampleType": "ManagedSystem"
      }
    ]   ← End of array of samples
  }
}
```

JSON format info and samples (watts)

```
{
  "systemUtil": {
    "utilInfo": {        ← one header
      "frequency": 30,
      "metricArrayOrder": [ "AVG"], ← Processed only has averages
      "metricType": "Processed",
      "mtms": "8284-22A*215296V",
       "name": "P8-S822-lime",          ← access by systemUtil[utilInfo][name]
      "uuid": "6ed2156b-342c-38cc-93d2-e01dad3c3fc7",
      "version": "1.3.0"
    },
    "utilSamples": [   ← long array of samples each with Watts + Celsius
      {                                  for sample in data[systemUtil][utilSamples]
        "energyUtil": {
          "powerUtil": {
            "powerReading": [ 452]              ← Watts - 1 per sample
          },
→sample[energyUtil][powerUtil][powerReading][0]
. . .
```

JSON format - Temperatures

```
{
  "systemUtil": {
...
  },
  "utilSamples": [   ← long array of samples each with Watts + Celsius
    {                          for sample in data[systemUtil][utilSamples]
...
        "thermalUtil": {
            "baseboardTemperatures": [
              {
                "entityId": "Baseboard temperature sensors(42h)",
                "entityInstance": "1",
                "temperatureReading": [ 27 ] ← Celsius - many per sample
              },
→sample[thermalUtil][baseboardTemperatures][temperatureReading][0]
```

Same for cpu Temperatures and inlet Temperatures
Note:
"Baseboard "in the JSON but I use POWER term "planar" in the stats

JSON format - Temperatures

```
{
  "systemUtil": {
...
  },
  "utilSamples": [   ← long array of samples each with Watts + Celsius
    {                          for sample in data[systemUtil][utilSamples]
...
      "sampleInfo": {← one at the end of the sample
          "status": 0,          ← Zero is good
          "timeStamp": "2017-05-25T15:00:30+0100"
        },
→sample[sampleInfo][timeStamp]
        "sampleType": "ManagedSystem"
```

Extract data from JSON format

```
>>> ret = requests.get(JSON_file_url, headers=head, verify=False)
...
data = json.loads(ret.text)                    ← load string in to JSON object called "data"
name = data['systemUtil']['utilInfo']['name']
mtms = data['systemUtil']['utilInfo']['mtms']

for sample in data['systemUtil']['utilSamples']:
   if sample['sampleInfo']['status'] != 0:
      print('Data Error')
   watts  = sample['energyUtil']['powerUtil']['powerReading'][0]
   mb0    = sample['energyUtil']['thermalUtil']['baseboardTemperatures'][0]['temperatureReading'][0]
   mb1    = sample['energyUtil']['thermalUtil']['baseboardTemperatures'][1]['temperatureReading'][0]
   mb2    = sample['energyUtil']['thermalUtil']['baseboardTemperatures'][2]['temperatureReading'][0]
   cpu0   = sample['energyUtil']['thermalUtil']['cpuTemperatures'][0]['temperatureReading'][0]
   cpu1   = sample['energyUtil']['thermalUtil']['cpuTemperatures'][1]['temperatureReading'][0]
   cpu2   = sample['energyUtil']['thermalUtil']['cpuTemperatures'][2]['temperatureReading'][0]
   cpu3   = sample['energyUtil']['thermalUtil']['cpuTemperatures'][3]['temperatureReading'][0]
   inlet  = sample['energyUtil']['thermalUtil']['inletTemperatures'][0]['temperatureReading'][0]
   timeStamp = sample['sampleInfo']['timeStamp']
```

And we are done

a) Save the stats to a CSV file: Server-name, date+time and the stats

b) Remove duplicate rows in the CSV (as the data overlaps)

c) Generate the graphs – simple text manipulation egoo.py

The SSP I/O is VERY similar

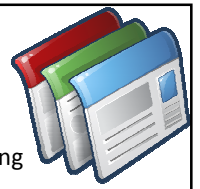
a) Get Preferences URL is different get back a list of SSP's (not Servers)

b) Same list of file names and fetching JSON data

c) The Stats are massive
   Ignore the disk level as its impossible to graph
   Ignore the tier level - could be added

d) The VIOS stats are a little tricky (variable number)
   The CSV file include a header line with the VIOS names
   used to create the graphs

Bugs: Documentation



- **The documentation was initially missing!**
- Then dozens of assumptions made and vital facts missing

- A working example is better than 1000 pages of documentation ☺
- I have covered many missing details and facts here!

- New improved Documentation on Friday 23rd June 2017

KnowledgeCenter URLs

https://www.ibm.com/support/knowledgecenter/P8ESS/p8ehl/concepts/ApiOverview.htm

https://www.ibm.com/support/knowledgecenter/P8ESS/p8ehl/apis/ManagedSystemPCMPreferences.htm

https://www.ibm.com/support/knowledgecenter/P8ESS/p8ehl/apis/EnergyMonitoring.htm

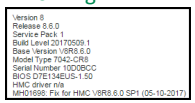
https://www.ibm.com/support/knowledgecenter/P8ESS/p8ehl/apis/SSPPreference.htm

Bugs: being fixed by development

Energy

•HMC FixPack1 = same data set returned every time despite new data available
- Work around - Fetch the stats **hourly** (or even slower)

• Possible bug: Latest HMC fix level stops the **E850** stats from working
  • Awaiting investigation by development



Shared Storage Pools – similar documentation issues
• Two fixes needed for VIOS collector to stop hanging
  • Raise a PMR requesting "VIOS perfprovider fixes"
• 1 in 7-ish records are full of zeros (work in progress)
  • These are silently removed in the goo.py Python program
• Stats not self consistent: Sum(VIOS) != whole SSP stats

To do list:

Energy

1. Twice changed format of XML & JSON files = my Python progs crashed
   • Stay tuned for updates via the AIXpert Blog – script to be more robust!
2. Need data from a 1 socket Server S812(L) or S814(L)
   and a E850 without the maximum sockets filled.
   • To check the number of sensors. Guessing: 3 planar + 2 chips + inlet = 6
   • Please, run the energy script with --debug & send me the JSON file

Shared Storage Pools

1. Reported I/O rates needs sanity checking with actual nmon stats
2. If you add/remove a VIOS from the SSP, the VIOS level graphs fail to display
   • Common CSV problem with an extra or "missing" column
   • JavaScript data arrays then fail to work at all
   • Best to remove the ssp_vios_io.csv file so to starts again

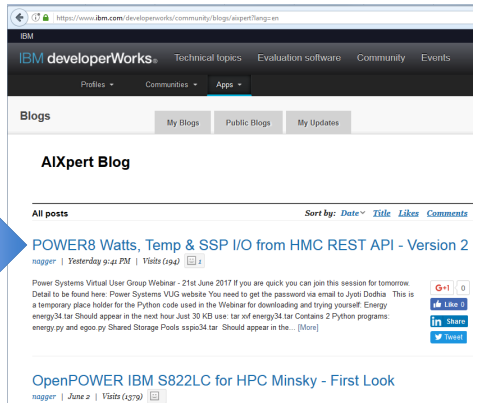
BOTH

• CSV dates: January is month zero! The way JavaScript likes it!
• Stats just build up to 1,000's – needs daily, weekly archive or something!

What next ?

1) Looking for **volunteers** to try out the Python programs + report back
2) AIXpert Blog
   https://www.ibm.com/developerworks/community/blogs/aixpert/entry/POWER8_Watts_Temp_SSP_I_O_from_HMC_REST_API_Version_2
3) Install Python, run and use CSV files or Googlechart graphs
4) Can enter at three levels
   a) Use the programs "as-is" and report back
      I can assist via debug output
   b) Use and improve/hack the programs to fit your needs
      Feedback your ideas
   c) Use them to "hack out" your own tool - I will answer questions
5) Help working through my To Do list or ideas.
   a) Robustness from data changes due to fixes!
   b) Data for 1 socket Scale-out like S812/S814
   c) VIOS's added or removed issue
   d) Getting swamped with data – weekly, monthly roll up

Google for:
AIXpert Blog



https://www.ibm.com/developerworks/community/blogs/aixpert/entry/POWER8_Watts_Temp_SSP_I_O_from_HMC_REST_API_Version_2