The Pixel Crate Monitor

Birk Engegaard, birk.engegaard@cern.ch / birkengegaard@gmail.com, July 2017

Content

- Introduction.
- How to run on a daily basis.
- Introduction to Grafana.
- Introduction to pix_cratemonitor.py.
- Improvements to be made.

Introduction

- Uses ipmitool commands in python2 to ask each crate in the pixel daq system for the sensor values of the cards in each crate.
- The monitoring script is run in 12 different cronjobs on the control hub pcs that correspond to the correct crate and in this cronjob added to the back end of Grafana.
- Sensors like temperatures, fan speeds, currents and voltages.
- The values are stored in a Graphite database and visualized in Grafana.
- The pix_cratemonitor.py script is run in a cronjob, but can be paused and resumed by running ./stop_pix_cratemonitor.py and ./start_pix_cratemonitor.py in /nfshome0/pixelpro/pix_cratemonitor/. It should be stopped temporarily when updating firmware to the FEDs and FECs.
- The script can exit with 3 different exit codes. 0 means everything is good, 1 is a warning and 2 is a critical error.
- Whenever the scripts exits with an exitcode other than 0 it will add a line in /nfshome0/pixelpro/pix_cratemonitor/pix_cratemon_errorlog.txt with information about what happened.

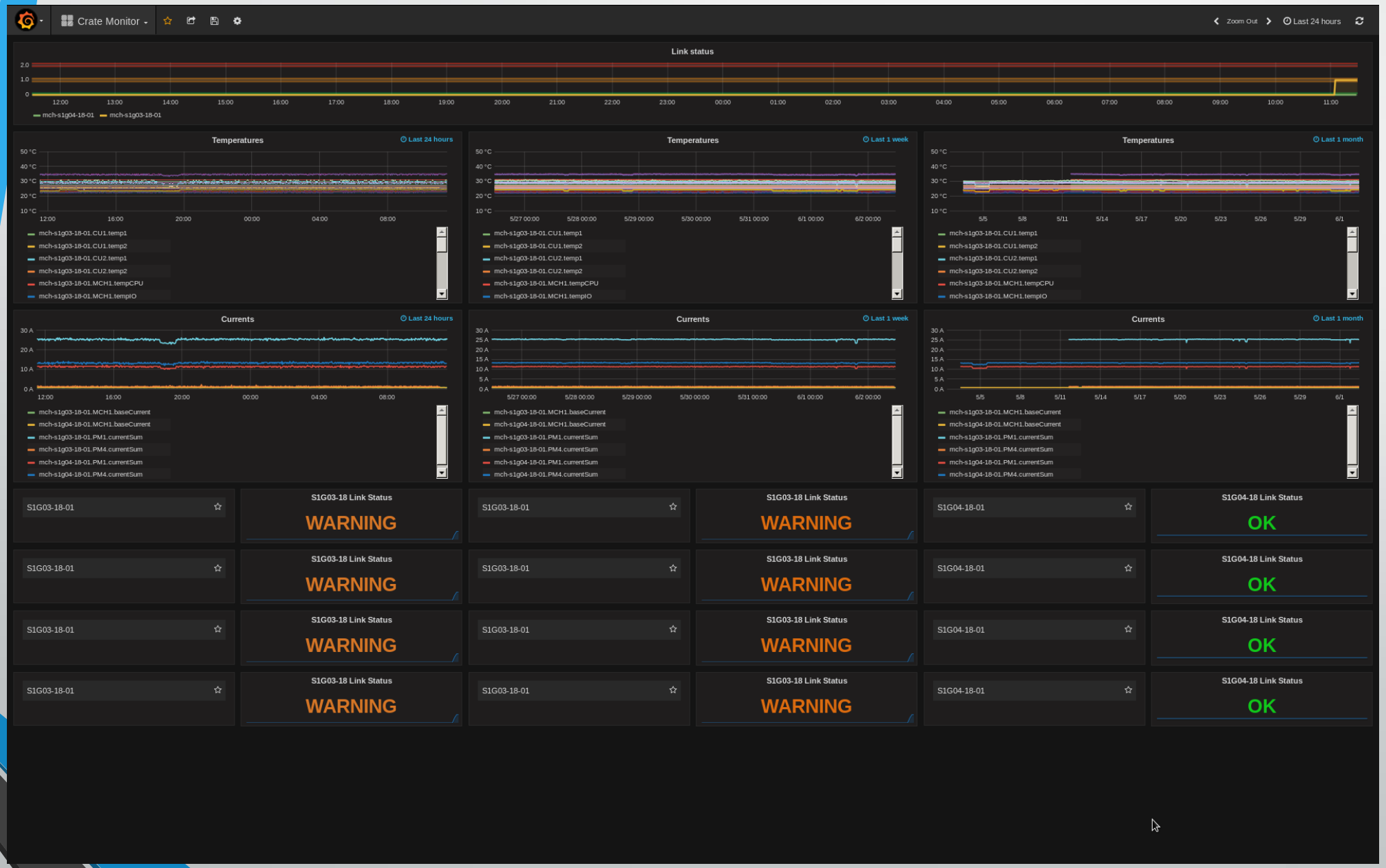
How to run on a daily basis

- The directory for using the Pixel Crate Monitor is "/nfshome0/pixelpro/pix_cratemonitor" wihtin the cms network.
- This folder has 4 files:
  - pix_cratemon_error.txt: A text file with info about errors.
  - Pix_cratemonitor_copy.py: A copy of the cratemonitor executable that is being held by the Grafana guys in the backend. If you want to improve the crate monitor you can run tests on this script and then submit a JIRA ticket to have the Grafana people push your change to the cronjobs.
  - start_pix_cratemon.py: run $ ./start_pix_cratemon.py to start the Pixel Crate Monitor.
  - stop_pix_cratemon.py: run $ ./stop_pix_cratemon.py to stop the Pixel Crate Monitor. This should be done when updating firmware etc. Remember to start it again.
  - run_status.txt: A text file that says either "Cratemon is running" or "Cratemon is stopped". The cratemon executable will read this file to determine whether it should ask the crates for sensor info or not. The start/stop files writes a line to this file.
- Contact person for the Grafana is Diego da Silva Gomes (diego.da.silva.gomes@cern.ch)

Introduction to Grafana

- The homepage of the Pixel Crate Monitor shows an overview of all the temperatures and currents the past day, week and months. It also shows the link status between Grafana and the crate
- On the top of the page you can see a graph of the link status, here you can see when a change in link status was triggered.
- What you're looking for on this page is changes in values. The variables are expected to have quite different values, but to remain stable
- On the bottom you see a list of all the crates with their link status. Clicking on one of the crates will bring you to another dashboard with more details about this crate like fan speeds and voltages
- The contact person for Grafana is Diego Da Silva Gomes (diego.da.silva.gomes@cern.ch)
- More general info about Grafana here https://grafana.com/

Grafana overview dashboard



Introduction to pix_cratemonitor.py

- Github: https://github.com/AstroSaiko/cratemonitor.
- Python2, uses ipmitool commands through subprocess().
- The script is general on crate level. That is the same script can be used for every crate. Run the script without an argument and it will figure out which MCH to talk to based on the controlhub it's being run on.
- The monitorable crate objects are represented as python classes with sensor variables as attributes.
- Upon instantiation the class will check card type and query it for it's data. If the type is unknown to the script it will return None values.
- The classes are easily modified to include more card types and brand now cards. The developer does not need to know everything in the script to modify it to work with a crate of choice.
- If the card does not yield any sensor values in the query cratemonitor.py will return None values for those sensors.
- Let's have a closer look at the classes in the script.

The MCH object

- Instantiation: MCH = MCH().
- Supported flavors: NAT-MCH-MCMC.
- Class functions: getData(), setHostname(), printSensorValues().
- getData() gives the sensor variables a value and returns a list of all the values.

```
class MCH:
    """MCH object"""
    def __init__(self, MCHIndex = 1):
        self.MCHIndex = MCHIndex # Some crates have multiple locations for MCHs
        self.entity = "194.{0}".format(str(96 + self.MCHIndex)) # converting MCH index to ipmi entity
        self.hostname = HOSTNAME # Global variable
        # Initializing empty variables
        self.flavor = None # MCH type
        self.tempCPU = None # CPU temperature
        self.tempIO = None # I/O temperature
        self.volt1V2 = None # 1.2V
        self.volt1V5 = None # 1.5V
        self.volt1V8 = None # 1.8V
        self.volt2V5 = None # 2.5V
        self.volt3V3 = None # 3.3V
        self.volt12V = None # 12V
        self.current = None # base current
        # Get data upon instantiation
        self.output = self.getData()
```

The PM object

- Instantiation: PM1 = PM(1), PM2 = PM(2).
- Supported flavors: NAT-PM-DC840.
- Class functions: getData(), setHostname(), printSensorValues().
- getData() gives the sensor variables a value and returns a list of all the values.

```
class PM:
    """Power module object"""
    def __init__(self, PMIndex):
        self.PMIndex = PMIndex # PM index in crate
        self.entity = "10.{0}".format(str(96 + self.PMIndex)) # converting PM index to ipmi entity
        self.hostname = HOSTNAME # Global variable
        # Initializing empty variables
        self.tempA = None # Temperature of brick-A
        self.tempB = None # Temperature of brick-B
        self.tempBase = None # Base temperature
        self.VIN = None # Input voltage
        self.VOutA = None # Output voltage A
        self.VOutB = None # Output voltage B
        self.volt12V = None # 12V
        self.volt3V3 = None # 3.3V
        self.currentSum = None # total current
        self.flavor = None # PM type
        # Get data upon instantiation
        self.output = self.getData()
```

The CU object

- Instantiation: CU1 = CU(1), CU2 = CU(2).
- Supported flavors: Schroff uTCA CU.
- Class functions: getData(), setHostname(), printSensorValues(), checkFlavor().
- getData() gives the sensor variables a value and returns a list of all the values.

```
class CU:
    '''Cooling Unit object'''
    def __init__(self, CUIndex):
        self.hostname = HOSTNAME # global variable
        self.CUIndex = CUIndex # CU index
        self.entity = "30.{0}".format(96 + CUIndex) # converting index to entity number
        if self.CUIndex == 1:
            self.target = "0xa8" # converting index to target ID
        else:
            self.target = "0xaa" # converting index to target ID
        # Initializing empty variables
        self.flavor = None # CU type
        self.CU3V3 = None # 3.3V
        self.CU12V = None # 12V
        self.CU12V_1 = None # 12V_1
        self.LM75Temp = None # temperature
        self.LM75Temp2 = None # temperature 2
        self.fan1 = None # fan speed
        self.fan2 = None # fan speed
        self.fan3 = None # fan speed
        self.fan4 = None # fan speed
        self.fan5 = None # fan speed
        self.fan6 = None # fan speed
        # Get data upon instantiation
        self.output = self.getData()
```

The AMC13 object

- Instantiation: amc13 = AMC13().
- Supported flavors: BU AMC13.
- Class functions: getData(), setHostname(), printSensorValues().
- getData() gives the sensor variables a value and returns a list of all the values.

```
class AMC13:
    '''AMC13 object'''
    def __init__(self):
        self.hostname = HOSTNAME # global variable
        # Initializing empty variables
        self.flavor = None # amc13 type
        self.T2Temp = None # T2 temperature
        self.volt12V = None # 12V
        self.volt3V3 = None # 3.3V
        self.volt1V2 = None # 1.2V
        # Get data upon instantiation
        self.output = self.getData()
```

What's being executed when the script is run

- This is what's being run when the script is executed.
- First the EXITCODE class will be instantiated to keep track of the link status. Then the crate objects will be instantiated and their variables filled with sensor values.
- The runstat variable tracks if cratemon is stopped or not. 0 means running, 1 means stopped.
- If the exitcode is 0 the script will print the sensorvalues with one formatting, and if it's something else the script will add an errormessage in it's output.
- The sensor info is formatted as sensorname1=value;;;; sensorname2=value;;;; for historical reasons, but is quickly changed to a format that the Grafana backend can understand.
- If the exitcode is nonzero the script will also run its errorMessage function which adds a line to the errorlog describing the event.
- Finally the script will exit with the correct exitcode.

```
if __name__ == "__main__":

    EXITCODE = EXITCODE() # For proper exit codes
    # Instantiate the objects in the crate

    # ====================================
    # Basic uTCA crate / Common for every crate
    # ====================================
    PM1 = PM(1)
    PM4 = PM(4)
    CU1 = CU(1)
    CU2 = CU(2)
    MCH = MCH()
    amc13 = AMC13()
    # ====================================
    # FC7s and crate specifics
    # ====================================

    # amc1 = FC7(1) # etc

    # ====================================
    # Format output
    # ====================================
    status = ["OK", "WARNING", "CRITICAL", "UNKNOWN"]
    runstat = 0 # Initiation run status as running
    if isStopped():
        runstat = 1 # cratemon is stopped/paused

    # Output for Icinga
    # if EXITCODE.getCode() == 0:
    #     print "Link status {0} | runstat={8};;;; linkStatus={7};;;; {1} {2} {3} {4} {5} {6}".format(status[EXITCODE.getCode()], PM1.output, PM4.output\
    #                                                                                                 , CU1.output, CU2.output, MCH.output, amc13.output, EXITCODE.getCode(), runstat)
    # else:
    #     print "Link status {0}, Message: {1} | runstat={9};;;; linkStatus={8};;;; {2} {3} {4} {5} {6} {7}".format(status[EXITCODE.getCode()], EXITCODE.getMsg(), PM1.output, PM4.output\
    #                                                                                                             , CU1.output, CU2.output, MCH.output, amc13.output, EXITCODE.getCode(), runstat)

    # Append errors to the error log
    if EXITCODE.getCode() != 0:
        errorMessage(EXITCODE.getMsg())

    # Output for Grafana / Graphite
    timestamp = time()
    output = "runstat={8};;;; linkStatus={7};;;; {1} {2} {3} {4} {5} {6}".format(status[EXITCODE.getCode()], PM1.output, PM4.output\
                                                                                , CU1.output, CU2.output, MCH.output, amc13.output, EXITCODE.getCode(), runstat, crate)

    output =  output.replace(';;;; ', '\n').replace('=', ' ').replace(';;;;', '').replace('_', '.').replace('.1.', '.1_').replace('.2.', '.2_').replace('.3.', '.3_').replace('.12V.1', '.12V_1').split('\n')
    for line in output:
        print 'cratemon.{0}.{1} {2}'.format(crate, line, timestamp)

    # Exit with appropriate exit code
    sys.exit(EXITCODE.getCode())
```

Improvements to be made

- Have the Grafana dashboards hosted on cmsonline.cern.ch
- Add FC7 functionality. The mmc code of the FC7s needs to be updated before ipmitool can understand the sensorinfo from the cards. The mmc code is being updated so this should be added at some point. A body for the FC7 class is already in the cratemonitor.py.
- I heard rumors that the pixelpilot user should be used instead of pixelpro. To make this change copy the pix_cratemonitor folder to /nfshome0/pixelpilot and change the paths in pix_cratemonitor.py, start_pix_cratemon.py and stop_pix_cratemon.py.
- Add functionality to clean the pix_cratemon_errorlog.txt if it becomes too large. It should take many, many years before this becomes an issue though.