```
#!/usr/bin/python3
import os
import json
import time
import sys
from datetime import datetime
from prometheus_client import Info,Counter,Gauge,Enum,start_http_server
from elasticsearch import Elasticsearch

# TODO: test for env
RESV_FILE = os.getenv('RESV_FILE', default=None)
NODE_FILE = os.getenv('NODE_FILE', default=None)
JOB_FILE = os.getenv('JOB_FILE', default=None)
scrape_interval = 30
port = 19998
es_host = '172.16.2.92'
es_port = 9200
es_jobs_index = 'slurm_jobs'
es_nodes_index = 'slurm_nodes'
es_timeout = '30s'
es = None

try:
    import pyslurm
except:
    print('Running without pyslurm')
    DEMO=True
    pass

# Slurm States
slurm_node_states = [ "allocated", "completing", "idle", "maint", "mixed", "perfctrs", "p
ower_up", "reserved" ]
slurm_useless_states = [ "reboot", "down", "drained", "draining", "fail", "failing", "fut
ure", "power_down", "unknown" ]
# Not built-in state
slurm_useless_states.append('not_responding')
# All states
slurm_node_states += slurm_useless_states
slurm_node_states_color = {
    "down" : 0, # Red
    "drained" : 5, # Orange foncÃ©
    "draining" : 10  # Green
}
# Job states
slurm_job_states = []
slurm_job_states_color = {
    "failed" : 0, # Red
    "pending" : 5, # Orange
    "running" : 10  # Green
}
slurm_login_nodes = [ "login-1", "login-2", "login-3", "login-4"]
slurm_partitions = ["unkillable","short-unkillable","main","main-grace","long","long-grac
e","cpu_jobs","cpu_jobs_low","cpu_jobs_low-grace"]

# States for CPU/GPU
slurm_pu_states=[ "alloc", "idle", "drain", 'total' ]
slurm_gpu_type=[]

# Filters
slurm_ignore_partitions=["gcpDebug", "cloud_tpux1Cloud", "gcpDevCPUCloud", "gcpMicroMemCP
UCloud", "gcpMiniMemCPUCloud", "gcpComputeCPUCloud", "gcpGeneralCPUCloud", "gcpMemoryCPUC
loud", "gcpV100x1Cloud", "gcpV100Xx1Cloud", "gcpV100x2Cloud", "gcpV100x4Cloud", "gcpK80x1
Cloud", "gcpK80Xx1Cloud", "gcpK80x2Cloud", "gcpK80x4Cloud"]

# CPU/GPU state counters
cpu_state_counters = {}
gpu_state_counters = {}
for pu_state in slurm_pu_states:
    cpu_state_counters[pu_state] = Gauge('slurm_cpus_' + pu_state, pu_state + ' CPUs', la
belnames=[ 'reservation' ])
```

```
    gpu_state_counters[pu_state] = Gauge('slurm_gpus_' + pu_state, pu_state + ' GPUs', la
belnames=[ 'type', 'reservation' ])

# Dict of nodes for state (use Enum ?)
node_states = {}
prom_node_states = Enum('slurm_node_states', 'States of nodes', states=slurm_node_states,
 labelnames=['name'])

# Job state counter
job_state_counter = Gauge('slurm_job_states', 'Slurm Job States', labelnames=['name'])

# Alloc node counter
login_node_counter = Gauge('slurm_login_nodes', 'Slurm Login Nodes', labelnames=['name'])

# Partitions counter
partition_counter = Gauge('slurm_partitions', 'Slurm Partitions', labelnames=['name'])

# Job date fields to convert
job_date_fields = [ 'submit_time', 'start_time', 'end_time', 'eligible_time']

# Reservation array
slurm_resv = {}

def xstr(s):
    if s is None:
        return ''
    return str(s)

def intersection(lst1, lst2):
    lst3 = [value for value in lst1 if value in lst2]
    return lst3

def sinfo():

    # Timestamp
    global unix_now, now, utc_now
    unix_now = time.time()
    now = datetime.now()
    utc_now = datetime.utcnow()
    now_pp = now.strftime("%d/%m/%Y %H:%M:%S")

    # Reset Gauge
    job_state_counter._metrics.clear()
    login_node_counter._metrics.clear()
    partition_counter._metrics.clear()
    for pu_state in slurm_pu_states:
        cpu_state_counters[pu_state]._metrics.clear()
        gpu_state_counters[pu_state]._metrics.clear()

    get_reservations()
    get_nodes()
    get_jobs()

    # Sleep
    print(now_pp + " Slurm stats generated")
    time.sleep(scrape_interval)

########################
def get_reservations():
    global slurm_resv
    slurm_resv = {
        "all"   : [],
        "nodes" : {}
    }

    if RESV_FILE is not None:
        with open(RESV_FILE, 'r') as json_file:
            resv_data = json.load(json_file)
    else:
        resv_data = pyslurm.reservation().get()
```

```
    for resv_name in resv_data:
        resv = resv_data[resv_name]
        # Test start and stop time
        if unix_now >= resv['start_time'] and unix_now <= resv['end_time'] :
            slurm_resv["all"].append(resv)
            # Get hostlist
            hl = pyslurm.hostlist()
            hl.create(resv['node_list'])
            res_nodes = hl.get_list()
            for res_node in res_nodes:
                if res_node not in slurm_resv["nodes"].keys():
                    slurm_resv["nodes"][res_node.decode('UTF-8')] = resv_name


##################
def get_nodes():

    # Build ES bulk body
    es_nodes_body = []

    if NODE_FILE is not None:
        with open(NODE_FILE, 'r') as json_file:
            nodes_data = json.load(json_file)
    else:
        nodes_data = pyslurm.node().get()

    for node_name in nodes_data:
        node = nodes_data[node_name]

        if node_name in slurm_resv["nodes"].keys():
            node["reservation"] = slurm_resv["nodes"][node_name]
        else:
            node["reservation"] = "None"

        ##########################################
        # Filters
        if(len(intersection(node['partitions'], slurm_ignore_partitions))>0):
            continue
        ##########################################

        # Add es action for each job
        es_nodes_body.append({'index': {'_id': node['name']}})

        # Split if node has 2 states
        if '+' in node['state']:
            multi = False
            node['state'] = node['state'].split("+")
            # Keep only 1 state DOWN>DRAIN>REST
            # UGLY: TODO better
            for _s in node['state']:
                if _s.startswith('DOWN'):
                    multi = "DOWN"
                    break
                elif _s.startswith('DRAIN'):
                    # DRAIN assumes DRAINED
                    multi = "DRAINED"
            if not multi:
                multi = node['state'][0]
            # Save
            node['state'] = multi

        # Count not responding nodes separatly
        if node['state'].endswith('*'):
            node['state'] = node['state'][:-1]
        # TODO: count reboots ?
        if node['state'].endswith(('#', '@')):
            node['state'] = node['state'][:-1]

        node['state'] = node['state'].lower()
```

```
        # Grafana wants UTC
        node['timestamp'] = utc_now

        # Boolean for state, only for Grafana colors
        if node['state'] in slurm_node_states_color.keys():
            node['node_state_code'] = slurm_node_states_color[node['state']]
            if node['reason_time'] is not None:
                node['reason_time_str'] = datetime.fromtimestamp(node['reason_time']).str
ftime('%Y-%m-%d %H:%M:%S')

        # Full body
        es_nodes_body.append(node)

        # Save in dict
        prom_node_states.labels(name=node['name']).state(node['state'])

        ##########################################
        # CPUS
        # save Idle CPUs to reuse
        node['idle_cpus'] = node['cpus'] - node['err_cpus']
        cpu_state_counters['total'].labels(reservation=node["reservation"]).inc(node['cpu
s'])
        cpu_state_counters['alloc'].labels(reservation=node["reservation"]).inc(node['all
oc_cpus'])
        cpu_state_counters['drain'].labels(reservation=node["reservation"]).inc(node['err
_cpus'])
        if node['state'] not in slurm_useless_states:
            cpu_state_counters['idle'].labels(reservation=node["reservation"]).inc(node['
idle_cpus'])
        else:
            cpu_state_counters['drain'].labels(reservation=node["reservation"]).inc(node[
'idle_cpus'])

        ##########################################
        # GPUs
        if 'gres' in node and len(node['gres'])>0:
            node['gpus'] = {}
            # Loop on Gres
            for g_tot in node['gres']:
                g_tot = g_tot.split(':',2)
                if len(g_tot) == 3:
                    node['gpus'].setdefault(g_tot[1], {})['total'] = int(g_tot[2].split('
(')[0])
            # Loop on Gres used
            for g_used in node['gres_used']:
                g_used = g_used.split(':',2)
                # Exclude MPS for now
                if len(g_used) == 3:
                    # Remove IDX index
                    node['gpus'].setdefault(g_used[1], {})['alloc'] = int(g_used[2].split
('(')[0])
            # Loop on sanitized gpus and set counters
            for gpu in node['gpus']:
                # Add to static
                if gpu not in slurm_gpu_type:
                    slurm_gpu_type.append(gpu)

                gpu_state_counters['total'].labels(type=gpu, reservation=node["reservatio
n"]).inc(node['gpus'][gpu]['total'])
                gpu_state_counters['alloc'].labels(type=gpu, reservation=node["reservatio
n"]).inc(node['gpus'][gpu]['alloc'])
                # Test state: if node is available/mixed and a least 1 cpu is avail, gres
 are idle otherwise drain
                if node['state'] not in slurm_useless_states and node['idle_cpus']>1:
                    gpu_state_counters['idle'].labels(type=gpu, reservation=node["reserva
tion"]).inc(node['gpus'][gpu]['total'] - node['gpus'][gpu]['alloc'])
                else:
                    gpu_state_counters['drain'].labels(type=gpu, reservation=node["reserv
ation"]).inc(node['gpus'][gpu]['total'] - node['gpus'][gpu]['alloc'])
```

```
    # Send bulk update
    es.bulk(index=es_nodes_index,  body=es_nodes_body, timeout=es_timeout)


#################
def get_jobs():
    if JOB_FILE is not None:
        with open(JOB_FILE, 'r') as json_file:
            jobs_data = json.load(json_file)
    else:
        jobs_data = pyslurm.job().get()

    # Build ES bulk body
    es_jobs_body = []
    for job in jobs_data:
        job_state = jobs_data[job]['job_state'].lower()

        # Add to static
        if job_state not in slurm_job_states:
            slurm_job_states.append(job_state)

        # Add es action for each job
        es_jobs_body.append({'index': {'_id': int(job)}})
        # Grafana wants UTC
        jobs_data[job]['timestamp'] = utc_now
        # Boolean for state, only for Grafana colors
        if job_state in slurm_job_states_color.keys():
            jobs_data[job]['job_state_code'] = slurm_job_states_color[job_state]

        # Create display time
        for date_field in job_date_fields:
            jobs_data[job][date_field + '_str'] = datetime.fromtimestamp(jobs_data[job][d
ate_field]).strftime('%Y-%m-%d %H:%M:%S')

        # Full body
        es_jobs_body.append(jobs_data[job])

        # Job counters
        # job_state_counters[jobs_data[job]['job_state'].lower()].inc()
        job_state_counter.labels(name=job_state).inc()

        # Login node counters
        alloc_node = jobs_data[job]['alloc_node'].split('.')[0]
        # Don't count compute nodes which are sometimes used to submit
        if alloc_node in slurm_login_nodes:
            # login_node_counters[alloc_node].inc()
            login_node_counter.labels(name=alloc_node).inc()

        # Partition counter
        if jobs_data[job]['partition'] in slurm_partitions:
            # partition_counters[jobs_data[job]['partition']].inc()
            partition_counter.labels(name=jobs_data[job]['partition']).inc()

    # Send bulk update
    es.bulk(index=es_jobs_index,  body=es_jobs_body, timeout=es_timeout)


if __name__ == '__main__':
    # Start up the server to expose the metrics.
    start_http_server(port)
    # Connect to ES
    es = Elasticsearch([
        {'host': es_host, 'port': es_port,  'use_ssl': False}
    ])
    es.indices.create(index=es_jobs_index, ignore=400)
    while True:
        sinfo()
```