StableNet® WeatherMap Statistics

June 16, 2021

1 Introduction

This script adds statistics to Weather Maps when given certain parameters as input over a CSV file. We describe the form the input file has to be of and the script's workflow with an example. It is important to note that the titles of columns in the input CSV file may not differ from those in the example file. However, the order of the columns may vary.

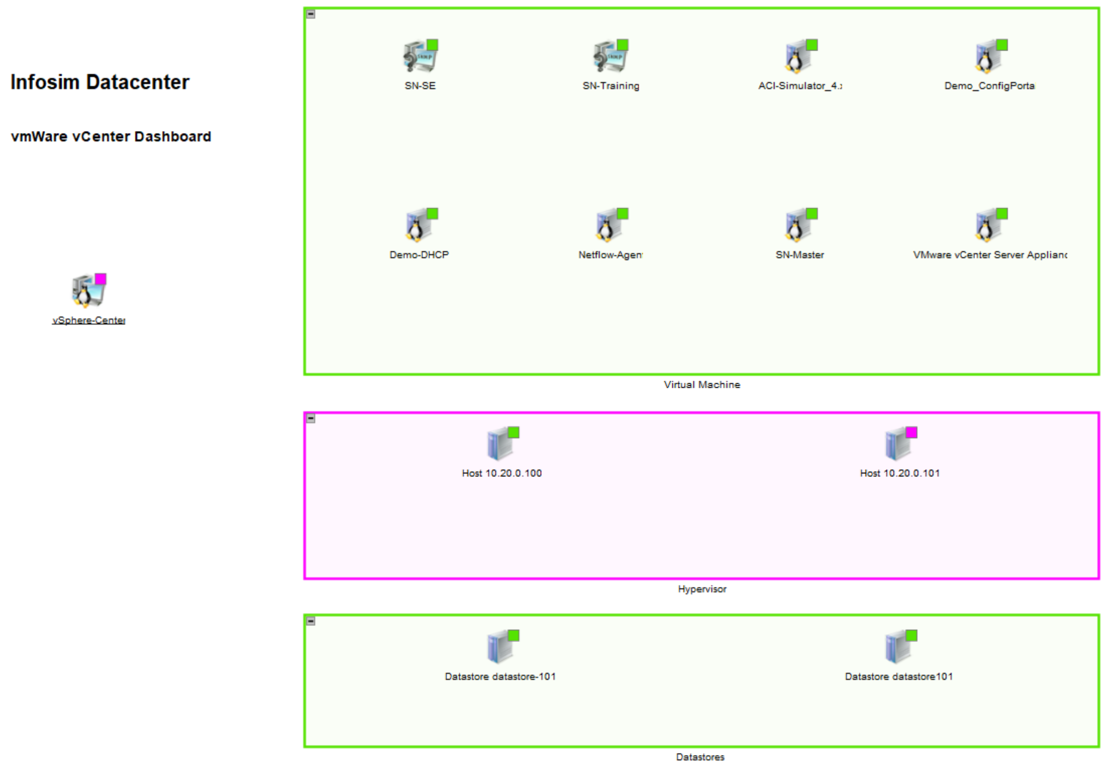
2 Example

For reasons of clarity, we refer to the external CSV file "input_node_trend.csv" instead of integrating it in to this file. Let us sketch the columns of the CSV files that are accepted as inputs of the current script. In most cases we only provide some exemplary entries. It can be made use of default values (as specified below) by leaving the corresponding entry empty. For a precise documentation we refer to the code.

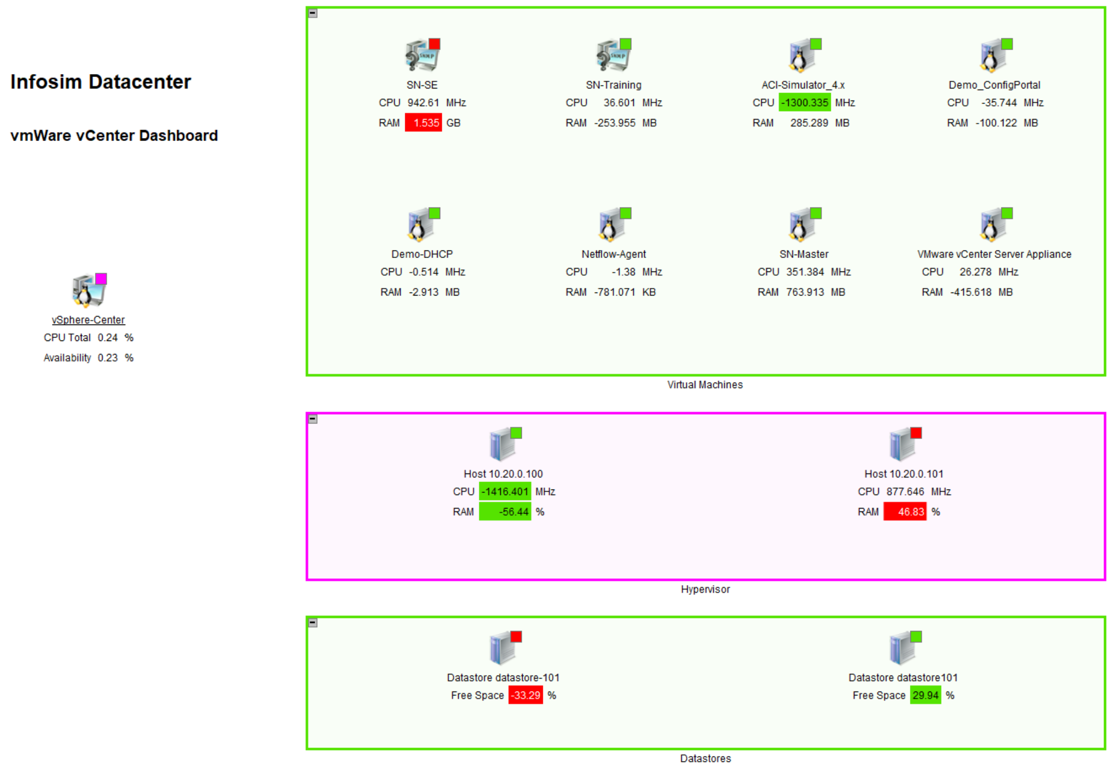
- measurement pattern: Here, a regular expression for the name of the measurement associated with the desired statistic can be specified. If a ping- or interface-measurement is required, enter "Ping measurement" or "Interface measurement". These measurements are not found by their names as the typically the names are the names of the corresponding devices or interfaces.
- aggregate: e.g., MAX, Q95, MEAN, COVERAGE, TREND. No regular expressions allowed. Case sensitive.
- metricname: e.g., CPU, Guest Memory, Free, total, System [U,u]ptime, t-octet. Note that here regular expressions may be used.
- metricunit: e.g., MB, Hz, %. Regular expressions allowed.
- lastvalue or measurementstat: Initial letters are sufficient.
- statistic title: As a default value, a standard name is chosen containing the names of the measurement and the metric.
- statistic ranges: The coloring of the statistics on the weathermap follows the entries provided here. A comma-separated list of strings of the form "lower_bound/upper_bound/color" can be provided, e.g., "90/180/500,180/Infinity/4000".
- statistic default state: typically 0, as the Ok-state is usually the default, i.e., if the entry is 0, the state is Ok if none of the "rules" under statistic ranges can be applied.

- showaslabel: Relevant for statistics on a Weather Map link. In case the entry is “true”, the statistic information is provided as label sticking to the link.
- metricscale add: All statistic values are increased by the value specified here. Default value: 0.
- metricscale multiply: All statistic values are multiplied by the value specified here. Default value: 1 (depending on the StableNet version you use you might want to change that to “100”).
- time multiplier: In this column and the next column the interval can be specified for which the statistic data are to be added.
- time type: e.g. lastmonths, lasthours, lastdays. Default value: lastmonths.
- offset multiplier: In this column and the next column an offset for the relevant time interval can be selected.
- offset type: Default value: lastmonths.
- time average
- node or link: initial letters suffice. Here it can be selected, whether the present line in the CSV file shall be added to a node or a link on the Weather Map.
- pattern for node name: Here a regular expression can be specified. In case “node” is chosen in the column just described, the statistics will only added to those Weather Map nodes that match this expression.
- source or destination: only relevant in case “link” has been selected in the column “node or link”. A Weather Map link contains three references: an elementreference, a sourcereference, and a destinationreference. By specifying “source” or “destination”, you can choose whether the script searches for measurements associated to the object referenced by sourcereference or destinationreference.
- domain: Initial three letters are sufficient. Possible values are “device”, “interface”, “measurement”, “monitor”, “module”, and “service”. Statistics are only added in case the referenced element has the specified domain.

Consider the following exemplary Weather Map.



When given the file "input_node_trend.csv" as input, the script constructs the following Weather Map.



vSphere-Center
CPU Total 0.24 %
Availability 0.23 %

SN-SE
CPU 942.61 MHz
RAM 1.535 GB

SN-Training
CPU 36.601 MHz
RAM -253.955 MB

ACI-Simulator_4.x
CPU -1300.335 MHz
RAM 285.289 MB

Demo_ConfigPortal
CPU -35.744 MHz
RAM -100.122 MB

Demo-DHCP
CPU -0.514 MHz
RAM -2.913 MB

Netflow-Agent
CPU -1.38 MHz
RAM -781.071 KB

SN-Master
CPU 351.384 MHz
RAM 763.913 MB

VMware vCenter Server Appliance
CPU 26.278 MHz
RAM -415.618 MB

Host 10.20.0.100
CPU -1416.401 MHz
RAM -56.44 %

Host 10.20.0.101
CPU 877.646 MHz
RAM 46.83 %

Datastore datastore-101
Free Space -33.29 %

Datastore datastore101
Free Space 29.94 %

3 Program

3.1 Imports and code definitions

3.1.1 Import necessary python modules

[ ]:
```
import warnings
import requests
from requests.auth import HTTPBasicAuth
import getpass
from xml.etree import ElementTree
import xml.dom.minidom # for pretty printing XML
import re # for regular expressions
import json
import csv
import sys
```

3.1.2 Function to normalize CSV entries

[ ]:
```
def normalized_entry(entry, i):
    if cols[i] == 'lastvalue or measurementstat':
        return 'lastvalue' if entry.lower().startswith('l') else␣
 ↪'measurementstat'
    if cols[i] == 'statistic default state':
        return entry if entry != '' else '0'
    if cols[i] == 'showaslabel':
        return 'true' if entry.lower().startswith('t') else 'false'
    if cols[i] == 'metricscale add':
        return entry if entry != '' else '0'
    if cols[i] == 'metricscale multiply':
        return entry if entry != '' else '1'
    if cols[i] == 'time multiplier':
        return entry if entry != '' else '1'
    if cols[i] == 'time type':
        return entry if entry != '' else 'lastmonths'
    if cols[i] == 'offset multiplier':
        return entry if entry != '' else '0'
    if cols[i] == 'offset type':
        return entry if entry != '' else 'lastmonths'
    if cols[i] == 'node or link':
        return 'node' if entry.lower().startswith('n') else 'link'
    if cols[i] == 'source or destination':
        if entry == '':
            return ''
        return 'source' if entry.lower().startswith('s') else 'destination'
    if cols[i] == 'domain':
        if entry.lower().startswith('d'):
            return 'device'
        if entry.lower().startswith('i'):
            return 'interface'
        if entry.lower().startswith('me'):
            return 'measurement'
        if entry.lower().startswith('mon'):
            return 'monitor'
        if entry.lower().startswith('mod'):
            return 'module'
        return 'service'
    return entry
```

3.1.3 Function to create and append statistic tag to Weather Map object (using <statistics>)

```
[ ]: def append_stat_tag():
         stat_attrs = {'metrickey': metric_key, 'type': stat_props['lastvalue or␣
      ↪measurementstat'],
                       'title': title, 'ranges': stat_props['statistic ranges'],
                       'defaultstate': stat_props['statistic default state'],
                       'showaslabel':stat_props['showaslabel']}
         statistic = ElementTree.SubElement(
             el.find('statistics'), 'statistic', stat_attrs
         )
         ElementTree.SubElement(
             statistic, 'reference', {'obid': meas_id, 'domain': 'measurement'}
         )
         ElementTree.SubElement(
             statistic, 'metricscale',
             {
                 'add': stat_props['metricscale add'],
                 'multiply': stat_props['metricscale multiply']
             }
         )
         if stat_props['lastvalue or measurementstat'] == 'measurementstat':
             ElementTree.SubElement(
                 statistic, 'time',
                 {
                     'multiplier': stat_props['time multiplier'],
                     'type': stat_props['time type'],
                     'average': stat_props['time average'],
                     'offsetmultiplier': stat_props['offset multiplier'],
                     'offsettype': stat_props['offset type']
                 }
             )
```

3.1.4 Functions to obtain the measurement for given Weather Map object (using /rest/tag/query)

```
[ ]: def get_andtagfilter_with_two_valuetagfilters(cat1, val1, cat2, val2):
         return '<andtagfilter>\
                     <valuetagfilter filtervalue="{}">\
                         <tagcategory key = "{}"/>\
                     </valuetagfilter>\
                     <valuetagfilter filtervalue="{}">\
                         <tagcategory key = "{}"/>\
                     </valuetagfilter>\
                 </andtagfilter>'.format(val1, cat1, val2, cat2)

     def get_andtagfilter_with_valuetagfilter_and_patterntagfilter(cat1, val, cat2,␣
      ↪pat):
         return '<andtagfilter>\
                     <valuetagfilter filtervalue="{}">\
                         <tagcategory key = "{}"/>\
                     </valuetagfilter>\
                     <patterntagfilter filterpattern="{}">\
                         <tagcategory key = "{}"/>\
                     </patterntagfilter>\
                 </andtagfilter>'.format(val, cat1, pat, cat2)

     def compute_measurement():
         url = 'https://{}:{}/rest/tag/query'.format(server_ip, server_port)
         filter = ''
         if obj_domain == 'device':
             #Here we consider Ping measurements separately because
             #the name of the measurement typically is the device name
             if stat_props['measurement pattern'] == 'Ping measurement':
                 filter +=  get_andtagfilter_with_two_valuetagfilters(
                     'Device ID', obj_id, 'Measurement Type', 'Ping')
             else:
                 filter += get_andtagfilter_with_valuetagfilter_and_patterntagfilter(
                     'Device ID', obj_id, 'Measurement Name', stat_props['measurement␣
      ↪pattern'])
         if obj_domain == 'interface':
             #Here we consider SNMP Interface measurements separately because
             #the name of the measurement typically is the device name
             if stat_props['measurement pattern'] == 'Interface measurement':
                 filter += get_andtagfilter_with_two_valuetagfilters(
                     'Interface ID', obj_id, 'Measurement Type', 'SNMP Interface')
             else:
                 filter += get_andtagfilter_with_valuetagfilter_and_patterntagfilter(
                     'Interface ID', obj_id, 'Measurement Name',␣
      ↪stat_props['measurement pattern'])
```

```
    if obj_domain == 'measurement':
        filter += get_andtagfilter_with_valuetagfilter_and_patterntagfilter(
            'Measurement ID', obj_id, 'Measurement Name',␣
↪stat_props['measurement pattern'])
    if obj_domain == 'monitor':
        filter += get_andtagfilter_with_valuetagfilter_and_patterntagfilter(
            'Monitor ID', obj_id, 'Measurement Name', stat_props['measurement␣
↪pattern'])
    if obj_domain == 'module':
        filter += get_andtagfilter_with_valuetagfilter_and_patterntagfilter(
            'Device Module ID', obj_id, 'Measurement Name',␣
↪stat_props['measurement pattern'])
    if obj_domain == 'service':
        filter += get_andtagfilter_with_valuetagfilter_and_patterntagfilter(
            'Service ID', obj_id, 'Measurement Name', stat_props['measurement␣
↪pattern'])
    query = '<taggablelistqueryinput domain="Measurement">\
                <tagcategories>\
                    <tagcategory key="Measurement ID"/>\
                    <tagcategory key="Measurement Name"/>\
                    </tagcategories>' +\
                filter +\
            '</taggablelistqueryinput>'
    resp = requests.post(
        url,
        data = query,
        verify = False,
        auth = HTTPBasicAuth(username, pw),
        headers = {'Content-Type': 'application/xml'}
    )
    meas = ElementTree.fromstring(resp.content)
    meas_id = ''
    meas_name = ''
    for element in meas.iter():
        if element.tag == 'tag':
            if element.get('key') == 'Measurement ID':
                meas_id = element.get('value')
            elif element.get('key') == 'Measurement Name':
                meas_name = element.get('value')
        if meas_id != '' and meas_name != '':
            break
    return (meas_id, meas_name)
```

3.1.5 Function to obtain metric key and metric name (making use of /rest/measurements/metric/{id})

```
[ ]: def compute_metric_key_and_name():
         resp = requests.get(
                 "https://" + server_ip + ":" + server_port +
                 "/rest/measurements/metric/" + meas_id,
                 verify=False,
                 auth=HTTPBasicAuth(username, pw)
             )
         metrics = ElementTree.fromstring(resp.content)
         metric_key = ''
         pat1 = re.compile(stat_props['metricname'])
         pat2 = re.compile(stat_props['metricunit'])
         for metric in metrics.iter():
             if pat1.search(str(metric.get('name'))):
                 if pat2.search(str(metric.get('unit'))):
                     if stat_props['lastvalue or measurementstat'] ==␣
      ↪'measurementstat':
                         metric_key = stat_props['aggregate'] + '_'
                     return (metric_key + metric.get('key'), metric.get('name'))
         return ('','')
```

3.1.6 Function to generate a standard statistic title (unless it is provided in the CSV file)

```
[ ]: def compute_statistic_title(input_title, meas_name, metric_name):
         title = ''
         if input_title != '':
             return input_title
         else:
             if stat_props['node or link'] == 'link':
                 title = 'Src ' if stat_props['source or destination'] == 'source'␣
      ↪else 'Dest '
             title += meas_name
             return title + metric_name
```

3.1.7 Function to request the Weather Map object as XML (using /rest/weathermaps/get/{id})

```
[ ]: def request_weathermap():
         url = "https://{}:{}/rest/weathermaps/get/{}".format(server_ip, server_port,␣
      ↪wmap_id)
         resp = requests.get(
             url,
             verify=False,
             auth=HTTPBasicAuth(username, pw)
         )
```

```
    #print (xml.dom.minidom.parseString(resp.content.decode('utf-8')).
↪toprettyxml())
    wmap = ElementTree.fromstring(resp.content)
    if wmap.tag == 'error':
        print('weathermap with id {} does not exist on server {}:{}'\
            .format(wmap_id, server_ip, server_port))
        sys.exit()
    # if flag is set, delete all existing statistics
    if delete_existing_stats:
        for el in wmap.iter():
            if el.tag == 'statistics':
                el.clear()
    return wmap
```

3.1.8 Function to check whether current object is relevant for the current line of the CSV file

[ ]:
```
def relevance_check():
    if obj_domain != stat_props['domain']:
        return False
    #test whether the name of the weathermapnode matches stat_props['pattern for␣
↪node name']
    if stat_props['pattern for node name'] == '' or el.tag != 'weathermapnode':
        return True
    pattern = re.compile(stat_props['pattern for node name'])
    if pattern.search(str(el.get('name'))) is None:
        return False
    return True
```

3.2 Actual program code to handle Weather Maps

Select this cell and run in menu: "Run > Run All Above Selected Cell"

3.2.1 Provide server credentials

[ ]:
```
#Credentials of server
server_ip = '10.20.20.113'
server_port = '5443'
username = 'infosim'
pw=getpass.getpass('Enter password for user ' + username + ' on server A:')
```

3.2.2 Check server credentials and get List of Weather Maps from Server

[ ]:
```
warnings.filterwarnings("ignore")
resp = requests.get(
    "https://"+server_ip+":"+server_port+"/rest/weathermaps/list",
    verify=False,
    auth=HTTPBasicAuth(username, pw)
)
tree = ElementTree.fromstring(resp.content)
if tree.tag == 'html':
    print('wrong credentials inserted')
    sys.exit()
for wmap in tree:
    wmap_name = wmap.get('name') if wmap.get('name') is not None else ''
    print('WeatherMap ' + wmap.get('obid') + ': ' + wmap_name)
```

3.2.3 Define input parameters for script

[ ]:
```
delete_existing_stats = True#if True, all existing statistics are deleted from␣
↪the weathermap
wmap_suffix = '_TREND';
wmap_id = '1058'
csv_file_name = 'input_node_trend.csv'
```

3.2.4 Read in statistic configuration from CSV file

[ ]:
```
inputs = []
cols = []
with open(csv_file_name, newline='') as csvfile:
    reader = csv.reader(csvfile, delimiter=';', quotechar='\'')
    first_line = True
    for row in reader:
        if not first_line:
            inputs += [{}]
            for i in range(0, len(row)):
                inputs[-1][cols[i]] = normalized_entry(row[i], i)
        else:
            for entry in row:
                cols += [entry]
            first_line = False
        if len(row) != len(cols):
            print('Malformed csv file: '
                  'not all lines of same length')
            break
for j in range(0, len(inputs)):
    print('Line: ' + str(j+1))
    for i in range(0, len(cols)):
        print('\t\t' + cols[i] + ': ' + inputs[j][cols[i]])
```

3.2.5 Add statistics to Weather Map XML and post it to server

```
[ ]: wmap = request_weathermap() # Request Weather Map with wmap_id from server␣
      ↪previously defined
     i = 0
     for stat_props in inputs:
         i += 1
         print('Processing line {} of input file'.format(str(i)))
         for el in wmap.findall('weathermapnodes/weathermapnode')\
             + wmap.findall('weathermaplinks/weathermaplink'):
             reference_tag = 'elementreference'
             if stat_props['node or link'] == 'link':
                 reference_tag = 'sourcereference' if stat_props['source or␣
      ↪destination'] == 'source'\
                         else 'destinationreference'
             obj_ref = el.find(reference_tag)
             if not hasattr(obj_ref, 'get'):
                 continue
             obj_id = obj_ref.get('obid')
             obj_domain = obj_ref.get('domain')
             if not relevance_check():
                 continue
             (meas_id, meas_name) = compute_measurement()
             if meas_id == '' or meas_name == '':
                 print('{:15}'.format(obj_domain + ' ' + obj_id) + 'Requested␣
      ↪measurement not found')
                 continue
             (metric_key, metric_name) = compute_metric_key_and_name()
             if metric_key == '':
                 print('{:15}'.format(obj_domain + ' ' + obj_id) + ': Requested␣
      ↪metric not found')
                 continue
             title = compute_statistic_title(stat_props['statistic title'],meas_name,␣
      ↪metric_name)
             append_stat_tag()
             print('{:15}'.format(obj_domain + ' ' + obj_id) + ' Statistic added')
     wmap.set('name', wmap.get('name') + wmap_suffix)
     #print(xml.dom.minidom.parseString(ElementTree.tostring(wmap)).toprettyxml())
     resp = requests.post(
         "https://" + server_ip + ":" + server_port + "/rest/weathermaps/add/",
         verify = False,
         auth = HTTPBasicAuth(username, pw),
         data = ElementTree.tostring(wmap),
         headers = {'Content-Type': 'application/xml'}
     )
     #print (xml.dom.minidom.parseString(resp.content.decode('utf-8')).toprettyxml())
```