Deploy Prometheus in Docker: (recommend)

1. Make Dockfile:

FROM prom/prometheus

ADD prometheus.yml /etc/prometheus/

"Dockerfile"

1. Make prometheus.yml:

global:

scrape\_interval: 10s

evaluation\_interval: 10s

scrape\_configs:

- job\_name: prometheus-app

static\_configs:

- targets: ["10.1.22.11:9090"]

- job\_name: nv-ex1

static\_configs:

- targets: ["10.1.22.11:1234"]

"prometheus.yml"

1. Build image:

$ sudo docker build -t prometheus .

1. Run Prometheus:

$ sudo docker run -d -p 9090:9090 --name prometheus-nv $ prometheus

1. Terminate Prometheus:

$ sudo docker kill prometheus

1. Remove exited Prometheus container:

$ sudo docker rm -v $(sudo docker ps -a -q -f status=exited)

1. For more information please refer:

*Prometheus: Up & Running - Infrastructure and Application Performance Monitoring* (By Brian Brazil)

Deploy Prometheus in Ubuntu:

1. Download and add the GPG key:

$ wget https://s3-eu-west-1.amazonaws.com/deb.robustperception.io/41EFC99D.gpg | sudo apt-key add –

1. Update the repository and install Prometheus:

$ sudo apt-get update -y

$ $ sudo apt-get install prometheus prometheus-node-exporter prometheus-pushgateway

1. Start Prometheus and enable it to start on boot time:

$ sudo systemctl start Prometheus

$ sudo systemctl enable Prometheus

1. Stop Prometheus:

$ sudo systemctl stop prometheu

1. For more information please refer:

https://www.howtoforge.com/tutorial/monitor-ubuntu-server-with-prometheus/

Deploy Prometheus from Source: (not recommend)

1. Deploy Go version 1.12 or higher (You may also need many other enironments)
2. A: Use the Go tool to download and install the prometheus and promtool binaries into your GOPATH:

$ go get github.com/prometheus/prometheus/cmd/...

$ prometheus --config.file=your\_config.yml

B: Or clone the repository yourself and build using make:

$ mkdir -p $GOPATH/src/github.com/Prometheus

$ cd $GOPATH/src/github.com/Prometheus

$ git clone <https://github.com/prometheus/prometheus.git>

$ cd Prometheus

$ make build

$ ./prometheus --config.file=your\_config.yml

1. For more information please refer:

https://github.com/prometheus/prometheus

Deploy Grafana in docker:(recommend)

1. Run Grafana:

$ sudo docker run -d -p 3000:3000 grafana/Grafana

1. For more information please refer:

https://grafana.com/docs/installation/docker/

Deploy Grafana in Ubuntu:

1. Download package:

$ wget <https://dl.grafana.com/oss/release/grafana_6.2.5_amd64.deb>

$ sudo dpkg -i grafana\_6.2.5\_amd64.deb

1. Set APT repository:

$ sudo apt-get install -y software-properties-common

$ sudo add-apt-repository "deb https://packages.grafana.com/oss/deb stable main"

1. Run Grafana:

$ sudo service grafana-server start

1. Sop Grafana:

$ sudo service grafana-server stop

1. For more information please refer:

https://grafana.com/docs/installation/debian/

Deploy Grafana from Source: (not recommend)

1. Deploy NodeJS version 6 and higher (You may also need many other installation)
2. Download and install Grafana in $GOPATH:

$ export GOPATH=`pwd`

$ go get github.com/grafana/Grafana

1. Building the backend:

$ cd $GOPATH/src/github.com/grafana/Grafana

$ go run build.go setup

$ go run build.go build

1. Build the Frontend Assets:

$ npm install -g yarn

$ yarn install --pure-lockfile

$ yarn start

1. Run Grafana:

$ ./bin/grafana-server

1. For more information please refer:

https://grafana.com/docs/project/building\_from\_source/

Prometheus Instruction:

1. After deployed Prometheus, open browser and go to:

Host\_IP:9090 (example: 10.1.22.11:9090)

1. On the top bar go to “Status -> Targets” to check exporter status. If the **name is blue** and “State” is “**UP**”, the exporter is running and Prometheus is successfully connected to the exporter.

A screenshot of a cell phone

Description automatically generated

1. On the top bar go to “Graph” and in the “Expression” box type “nv” to view all the metrics the exporter has.

A screenshot of a cell phone

Description automatically generated

1. Click on the metric you want to see, then click “Execute” to get all the data in this metric.

A screenshot of a cell phone

Description automatically generated

1. Click on the “Graph” to get a sample graph.

A screenshot of a computer

Description automatically generated

1. Prometheus support some basic calculations to help analyzing the data. For more information you can find it here:

https://prometheus.io/docs/prometheus/latest/querying/basics/

A screenshot of a computer

Description automatically generated

Grafana Instruction:

1. After deployed Prometheus, open browser and go to:

Host\_IP:3000 (example: 10.1.22.11:3000)

1. Login with username: admin, password: admin:



1. Change password or skip.
2. Add data source:
   1. Click on the green “Add data source”
   2. Choose data type: Prometheus
   3. In the “HTTP -> URL”, put the Prometheus address in it:
      1. http://Host\_IP:9090 (example: <http://10.1.22.11:9090>)
   4. Change other settings if you like, then click the green “Save & Test”
   5. If it says “Data source is working”, click on the “Back”.
3. Import dashboard:
   1. Find the “+” on the left bar, select “Import”:

A screenshot of a cell phone

Description automatically generated

* 1. Upload our dashboard templet JSON file using the green bottom or copy the JSON file into the bigger box and load with the blue bottom.

A screenshot of a cell phone

Description automatically generated

* 1. You should get the dashboard and see data on it.

A screen shot of a computer

Description automatically generated

1. Dashboard cumtomization:
   1. You can select dashboard update period by clicking the Orange color number in the top-right corner. Or update mannully by clicking the “Cycle icon”. These time period can be set in “Dashboard settings (fifth icon top-right) -> General -> Time Options -> Auto-refresh”

A picture containing electronics

Description automatically generated

* 1. You can select data time interval by clicking the box with text.

A screenshot of a computer screen

Description automatically generated

* 1. Or go into a view mode of the dashboard using the monitor icon.

A screen shot of a video game

Description automatically generated

* 1. You can change the dashboard name and many other settings in the “dashboard settings”

A screenshot of a cell phone screen with text

Description automatically generated

* 1. Switch to other dashboards by clicking the dashboard name on the top-left.

A screenshot of a cell phone

Description automatically generated

* 1. You can drag every panel to rearrange them by clicking the top of it. And resize them by clicking the bottom-right.

A screen shot of a computer

Description automatically generated

* 1. Don’t forget to “Save” the dashboard after you change it.

1. Panel data
   1. Click on the “Add panel” on the top bar to add a panel:

A screenshot of a cell phone

Description automatically generated

* 1. Let’s start with “Add Query”. In the Query box marked with “A”, type “nv” and select the metric you want to put in this panel. Enforcers memory usage for example:

A display screen

Description automatically generated

* 1. After we put the metric in the query, we can see many data show up in the graph with very long name. The long names are combined by their metric name and labels added to them in the exporter. For our convenience, we want to shorten it and make it easy to read. For enforcers, we only want to display their enforcer names. So we can type “{{display}}” in the “Legend” box.

A close up of a screen

Description automatically generated

* 1. The labels are really convenient. Not only for displaying, we can also sort the data with labels. For example, if we only want to get the enforcers from certain one host, we can select data with the name of that host:

A close up of a screen

Description automatically generated

* 1. In the next section, “Visualization”, we can select the panel type we want to display the data. Let try “Singlestat”.
  2. The default data is in bytes, so we can set its unit in the “Visualization -> Value -> Unit” and choose “Data -> bytes”. Please notice that 1kb = 1024b in Data (IEC) and 1kb = 1000b in Data (Metric)

A screen shot of a video game

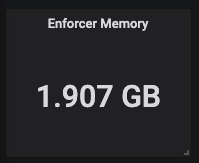
Description automatically generated

* 1. To view the current value, select “Current” in “value -> Stat” and also you can change color with your preference.
  2. Go to the “general” section to change the “Title” of your panel.

A screen shot of a smart phone

Description automatically generated

* 1. Go back to the dashboard we can see this panel:



1. Other settings: You can manage your account and add, remove, and edit other accounts in a “admin” account. There are three kinds of users: Admin, Editor, and Viewer. Viewers only have access to dashboard and can’t edit it. Editors can edit dashboards. Admin can manage users, dashboards, plugins and many other things.

Exporter Instruction:

1. Install Python3:

$ sudo apt-get install software-properties-common

$ sudo add-apt-repository ppa:deadsnakes/ppa

$ sudo apt-get update

$ sudo apt-get install python3.6

1. Install the Prometheus Python client:

$ sudo pip install -U setuptools

$ sudo pip install -U pip

$ sudo pip install prometheus\_client requests

1. Run exporter:

$ python3 nv\_exporter.py [port] [target\_ip:port]

(example: $ python3 nv\_exporter.py 1234 10.1.22.11:30443)

1. Open browser, go to: Host\_IP:port (example: 10.1.22.11:1234)
2. If you can load the metric page, the exporter is working fine.

A close up of a newspaper

Description automatically generated

Code Documentation:

1. Import libraries:

• from prometheus\_client import start\_http\_server, Metric, REGISTRY

• from datetime import datetime

• import json

• import requests

• import sys

• import time

• import urllib3

1. Disable insecure request warning:

• urllib3.disable\_warnings( urllib3.exceptions.InsecureRequestWarning)

1. In class apiCollector, split user input endpoints:

• self.\_endpoint = endpoint.split('+')

1. For each target user inputs, assign a for loop for the following functionalities:

• for target in self.\_endpoint:

1. Take out target address:

• ep1 = target.split(':')

• ep = ep1[0]

1. Log in and get token:

• data = '{"password": {"username": "admin", "password": "admin"}}'

• headers = {'Content-Type': 'application/json'}

• response = requests.post('https://' +target+ '/v1/auth', headers=headers, data=data, verify=False)

• token = json.loads(response.text)["token"]["token"]

• headers = {'Content-Type': 'application/json','X-Auth-Token': token}

1. Get system summary JSON:

• response = requests.get('https://' +target+ '/v1/system/summary', headers=headers, verify=False)

• sjson = json.loads(response.text)

1. Open a metric and set the data to Prometheus metric:

• metric = Metric('nv\_summary', 'A summary of ' +ep, 'summary')

• metric.add\_sample('nv\_summary\_services', value=sjson["summary"]["services"], labels={'target':ep})

• Etc.

1. For "cvedb\_create\_time", convert the format:

• dt = sjson["summary"]["cvedb\_create\_time"]

• dt1 = dt.split('-')#Year, Mon

• dt2 = dt1[2].split('T')#Day

• dt3 = dt2[1].split(':')#Hour, Min

• dt4 = dt3[2].split('Z')#Sec

• ap = 'AM'

• if int(dt3[0]) > 12:

• ap = 'PM'

• dt3[0] = int(dt3[0]) – 12

• time1 = dt1[1]+' '+dt2[0]+' '+dt1[0]+' '+dt3[0]+':'+dt3[1]+':'+dt4[0]+ap

1. Set system default start time to a variable:

• time0 = datetime.strptime('12 31 1969 5:00:00PM', '%m %d %Y %I:%M:%S%p')

1. Convert the time string in to milliseconds:

• time2 = datetime.strptime(time1, '%m %d %Y %I:%M:%S%p')

• diff = int((time2 - time0).total\_seconds()\*1000)

1. Add the "cvedb\_create\_time" to Prometheus metrics and close this metric.

• metric.add\_sample('nv\_summary\_cvedbTime', value=diff, labels={'target':ep})

• yield metric

1. Get conversation JSON:

• response = requests.get('https://' +target+ '/v1/conversation', headers=headers, verify=False)

1. Open a new metric for conversation:

• metric = Metric('nv\_conversation', 'conversation of ' +ep, 'gauge')

1. For each 'conversations' in the JSON file, check their ports:

• for c in json.loads(response.text)['conversations']:

• try: c['ports']

• except KeyError: port\_exists = False

• else: port\_exists = True

1. If the port exsit, add its conversation bytes to Prometheus metric:

• if port\_exists is True:

• for k in c['ports']:

• metric.add\_sample('nv\_conversation\_bytes', value=c['bytes'], labels={'port': k, 'from': c['from'], 'to': c['to'], 'target':ep})

• yield metric

1. Get enforcer JSON:

• response = requests.get('https://' +target+ '/v1/enforcer', headers=headers, verify=False)

1. Open a metric and for each enforcer in the JSON file, Get their status JSON file:

• metric = Metric('nv\_enforcer', 'enforcers of ' +ep, 'gauge')

• for c in json.loads(response.text)['enforcers']:

• response2 = requests.get('https://' +target+ '/v1/enforcer/' + c['id'] + '/stats', headers=headers, verify=False)

• ejson = json.loads(response2.text)

1. Add their CPU and memory data to metric and close the metric:

• metric.add\_sample('nv\_enforcer\_cpu', value=ejson['stats']['span\_1']['cpu'], labels={'id': c['id'], 'host': c['host\_name'], 'display': c['display\_name'], 'target':ep})

• metric.add\_sample('nv\_enforcer\_memory', value=ejson['stats']['span\_1']['memory'], labels={'id': c['id'], 'host': c['host\_name'], 'display': c['display\_name'], 'target':ep})

• yield metric

1. Get host JSON:

• response = requests.get('https://' +target+ '/v1/host', headers=headers, verify=False)

1. Set host metrics

• metric = Metric('nv\_host', 'host information of ' +ep, 'gauge')

• for c in json.loads(response.text)['hosts']:

• metric.add\_sample('nv\_host\_memory', value=c['memory'], • labels={'name': c['name'], 'id': c['id'], 'target':ep})

• yield metric

1. Get debug admission stats:

• response = requests.get('https://' +target+ '/v1/debug/admission\_stats', headers=headers, verify=False)

• djson = json.loads(response.text)

1. Set admission metrics:

• metric = Metric('nv\_admission', 'Debug admission stats of ' +ep, 'gauge')

• metric.add\_sample('nv\_admission\_allowed', value=djson['stats']['k8s\_allowed\_requests'], labels={'target':ep})

• metric.add\_sample('nv\_admission\_denied', value=djson['stats']['k8s\_denied\_requests'], labels={'target':ep})

• yield metric

1. Get image vulnerability JSON:

• response = requests.get('https://' +target+ '/v1/scan/registry', headers=headers, verify=False)

1. Set vulnerability metrics, for each registryin summary, request its image JSON:

• for c in json.loads(response.text)['summarys']:

• response2 = requests.get('https://' +target+ '/v1/scan/registry/' + c['name'] + '/images', headers=headers, verify=False)

1. For each image, add its high and medium value to the metric:

• for i in json.loads(response2.text)['images']:

• metric.add\_sample('nv\_image\_vulnerabilityHigh', value=i['high'], labels={'name': c['name'], 'imageid': i['image\_id'], 'target':ep})

• metric.add\_sample('nv\_image\_vulnerabilityMedium', value=i['medium'], labels={'name': c['name'], 'imageid': i['image\_id'], 'target':ep})

• yield metric

1. Get container vulnerability JSON:

• response = requests.get('https://' +target+ '/v1/scan/workload', headers=headers, verify=False)

1. Set vulnerability metrics, create a list for same container name:

• cvlist = []

• metric = Metric('nv\_container\_vulnerability', 'container vulnerability of ' +ep, 'gauge')

• for c in json.loads(response.text)['workloads']:

1. Merge containers with similar name and add them to metric:

• if c['service'] not in cvlist and c['service\_mesh\_sidecar'] is False and c['high']!=0 and c['medium']!=0:

• if ("-pod-" not in c['service'] and "default" not in c['service']) or "-pod-00" in c['service'] or "-v1" in c['service']:

• metric.add\_sample('nv\_container\_ vulnerabilityHigh', value=c['high'], labels={'service': c['service'], 'target':ep})

• metric.add\_sample('nv\_container\_ vulnerabilityMedium', value=c['medium'], labels={'service': c['service'], 'target':ep})

• cvlist.append(c['service'])

• yield metric

1. Set Security log metrics:

• metric = Metric('nv\_log', 'log of ' +ep, 'gauge')

1. Get threat log:

• response = requests.get('https://' +target+ '/v1/log/threat', headers=headers, verify=False)

1. Save data in 5 lists:

• ttimelist = []

• tnamelist = []

• tcnamelist = []

• tsnamelist = []

• tidlist = []

• for c in json.loads(response.text)['threats']:

• ttimelist.append(c['reported\_timestamp'])

• tnamelist.append(c['name'])

• tcnamelist.append(c['client\_workload\_name'])

• tsnamelist.append(c['server\_workload\_name'])

• tidlist.append(c['id'])

1. Save the latest 6 log into metric:

• for x in range(0,5):

• metric.add\_sample('nv\_log\_events', value=ttimelist[x]\*1000, labels={'log': "thread", 'fromname': tcnamelist[x], 'toname': " -> " +tsnamelist[x], 'id': tidlist[x], 'name': tnamelist[x], 'target':ep})

1. Get incident log:

• response = requests.get('https://' +target+ '/v1/log/incident', headers=headers, verify=False)

1. Create 4 lists:

• itimelist = []

• inamelist = []

• iwnamelist = []

• iidlist = []

1. Check incidents workload names:

• for c in json.loads(response.text)['incidents']:

• try: c['workload\_name']

• except KeyError: workload\_exists = False

• else: workload\_exists = True

1. If the workload exists, save data into lists:

• if workload\_exists is True:

• itimelist.append(c['reported\_timestamp'])

• inamelist.append(c['name'])

• iwnamelist.append(c['workload\_name'])

• iidlist.append(c['workload\_id'])

1. Save the latest 6 log into metric:

• for x in range(0,5):

• metric.add\_sample('nv\_log\_events', value=itimelist[x]\*1000, labels={'log': "incident", 'fromname': iwnamelist[x], 'toname': " ", 'name': inamelist[x], 'id': iidlist[x], 'target':ep})

1. Get violation log:

• response = requests.get('https://' +target+ '/v1/log/violation', headers=headers, verify=False)

1. Save data in 5 lists:

• vtimelist = []

• vnamelist = []

• vcnamelist = []

• vsnamelist = []

• vidlist = []

• for c in json.loads(response.text)['violations']:

• vtimelist.append(c['reported\_timestamp'])

• vcnamelist.append(c['client\_name'])

• vnamelist.append(c['cluster\_name'])

• vsnamelist.append(c['server\_name'])

• vidlist.append(c['client\_id']+c['server\_id'])

1. Add latest 6 log into metric and close metric:

• for x in range(0,5):

• metric.add\_sample('nv\_log\_events', value=vtimelist[x]\*1000, labels={'log': "violation", 'id': vidlist[x], 'toname': " -> " +vsnamelist[x], 'fromname': vcnamelist[x], 'name': vnamelist[x], 'target': ep})

• yield metric

1. Delete token and log out:

• response = requests.delete('https://' +target+ '/v1/auth', headers=headers, verify=False)

1. main:

• start\_http\_server(int(sys.argv[1]))

• REGISTRY.register(apiCollector(sys.argv[2]))

• while True: time.sleep(15)