Stock Market time series analysis using OpenStack, Gnocchi and Grafana

Cloud Computing final project - Prof. Carlo Vallati

Leonardo Turchetti
Lorenzo Tonelli
Ludovica Cocchella
Rambod Rahmani

Msc. in Artificial Intelligence and Data Engineering

May 23, 2021

Contents

1 Introduction

The project focused on deploying Gnocchi - an open-source time series database - and Grafana - an open source analytics and monitoring solution - on the preexisting OpenStack installation. After deployment, some preliminary tests were carried out in order to make sure everything was running smoothly. Finally, Gnocchi metrics were created and populated using stock market data. Grafana was then used to visualize the data and extract useful statistics.

In what follows, the deployment procedures and the ad-hoc required configurations are detailed. Just for reference, the preexisting OpenStack installation consists of:

```
1 172.16.3.218    Juju Controller
2 172.16.3.238    Compute Node 0/OpenStack Controller
3 172.16.3.177    Compute Node 1
```

```
4 172.16.3.174    Compute Node 2
5 172.16.3.227    Compute Node 3
```

The entire codebase is available at https://github.com/lorytony/CloudComputing.

2 Gnocchi Deployment

Gnocchi is an open-source time series database. The problem that Gnocchi solves is the storage and indexing of time series data and resources at a large scale. Gnocchi takes a unique approach to time series storage: rather than storing raw data points, it aggregates them before storing them. This built-in feature is different from most other time series databases, which usually support this mechanism as an option and compute aggregation (average, minimum, etc.) at query time.

The Gnocchi database was deployed[1] to compute node 0 in a container using Juju

```
$ juju deploy --to lxd:0 cs:gnocchi
$ juju deploy --to lxd:0 cs:memcached
$ juju add-relation gnocchi mysql
$ juju add-relation gnocchi memcached
$ juju add-relation gnocchi keystone
$ juju add-relation gnocchi ceph-mon
```

The choice was made to deploy Gnocchi inside a container. As a result, initially, Gnocchi was accessible only from the OpenStack controller node. For ease of debugging, port forwarding was applied on the OpenStack controller node in order to make Gnocchi visible from outside.

```
$ iptables -t nat -A PREROUTING -i eth0 -p tcp -m tcp --dport 5000 -j DNAT ...
    --to-destination 252.3.238.205:5000
$ iptables -t nat -A POSTROUTING -d 252.3.238.205/32 -o eth0 -j MASQUERADE

$ iptables -t nat -A PREROUTING -i eth0 -p tcp -m tcp --dport 8041 -j DNAT ...
    --to-destination 252.3.238.176:8041
$ iptables -t nat -A POSTROUTING -d 252.3.238.176/32 -o eth0 -j MASQUERADE
```

From within the OpenStack network and from outside, Gnocchi can now be accessed using the OpenStack controller IP address:

```
$ source admin-openrc.sh
$ openstack token issue
$ gnocchi --endpoint http://172.16.3.238:8041 metric list
```

Right after deployment, some preliminary tests - creating a metric, pushing and reading measurements - were performed using Gnocchi REST API to check if the deployment was successful.

2.1 Metrics

Gnocchi is based on the concept of metrics which contain measurements: as we intend to store stock market data, a new metric was created for each of the stocks we intend to monitor. The top 15 stocks of the Italian Stock Exchange were selected.

The official Gnocchi Python client was used to create the required metrics:

[1] https://jaas.ai/gnocchi/37

```
$ gnocchi metric create --archive-policy-name high "ATL.MI"
$ gnocchi metric create --archive-policy-name high "UCG.MI"
$ gnocchi metric create --archive-policy-name high "EXO.MI"
$ gnocchi metric create --archive-policy-name high "ISP.MI"
$ gnocchi metric create --archive-policy-name high "G.MI"
...
```

The default `archive-policy` used is `high`. By default, 4 archive policies are created. The name both describes the storage space and CPU usage needs.

- low
  - 5 minutes granularity over 30 days;
- medium
  - 1 minute granularity over 7 days;
  - 1 hour granularity over 365 days;
- high
  - 1 second granularity over 1 hour;
  - 1 minute granularity over 1 week;
  - 1 hour granularity over 1 year.

The archive policies define how the metrics are aggregated and how long they are stored. Each archive policy definition is expressed as the number of points over a timespan. Each archive policy also defines which aggregation methods will be used. The default is set to `default_aggregation_methods` which is by default set to `mean`, `min`, `max`, `sum`, `std`, `count`.

Both the archive policy and the granularity entirely depends on your use case. For our intended usage, the default `archive-policy high` and the default aggregations methods were considered more than enough. Be aware that the more definitions you set in an archive policy, the more CPU it will consume.

3 Grafana Deployment

Grafana is a multi-platform open source analytics and interactive visualization web application. It provides charts, graphs, and alerts for the web when connected to supported data sources. It is expandable through a plug-in system. End users can create complex monitoring dashboards using interactive query builders.

Grafana was deployed[2] to compute node 0 using Juju

```
$ juju deploy cs:grafana
```

The choice was made not to deploy Grafana inside a container in order to be able to easily access its web interface and avoid further configurations related to port forwarding.

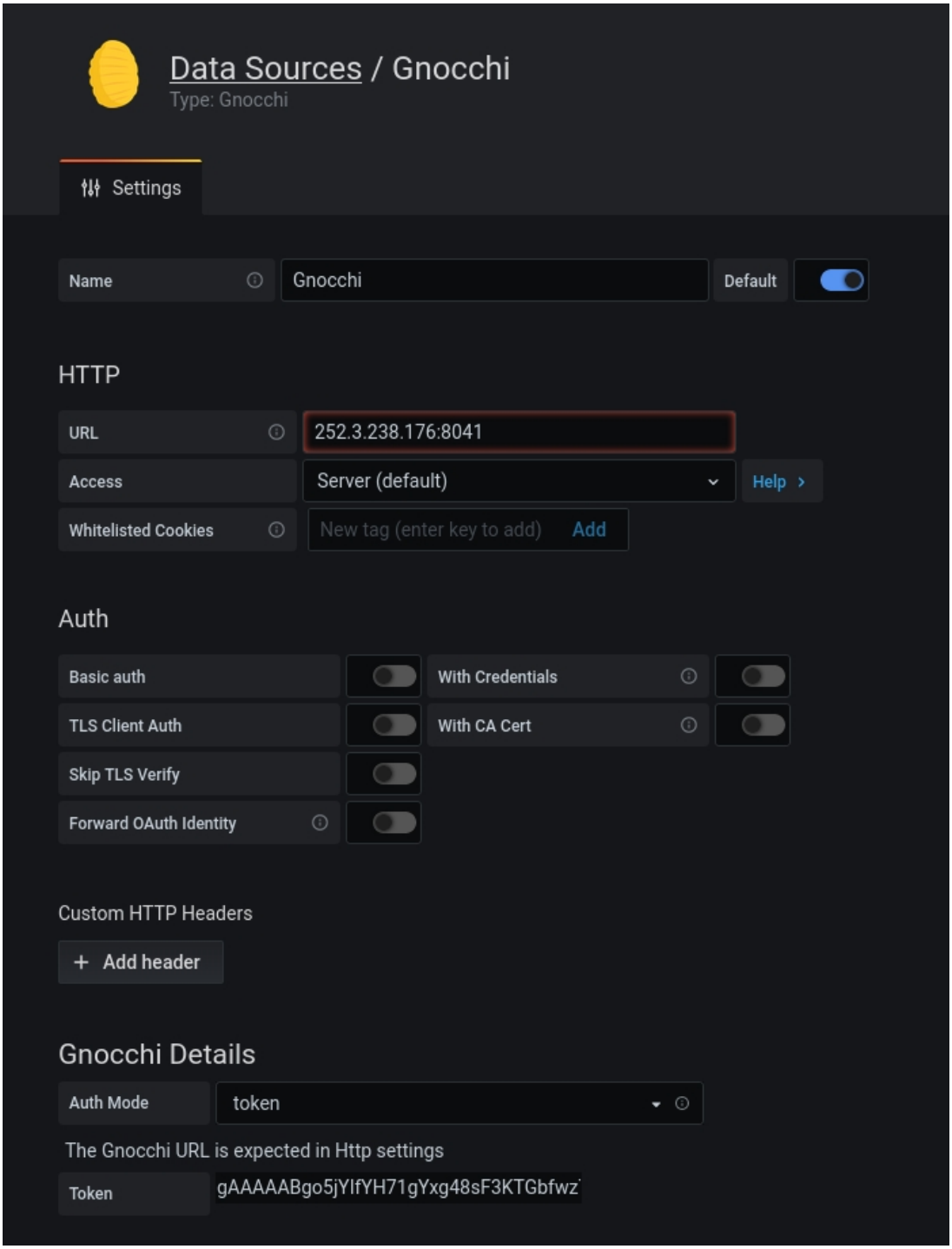
[2] https://jaas.ai/grafana/40

3.1 Gnocchi Datasource

The Gnocchi datasource was installed via grafana.net[3]:

```
$ grafana-cli plugins install gnocchixyz-gnocchi-datasource
$ systemctl restart grafana-server
```

Finally, a new Gnocchi data source was added using the Grafana web interface:



[3] https://grafana.com/grafana/plugins/gnocchixyz-gnocchi-datasource

As it can be seen in the previous picture, Grafana needs a Keystone token as part of the Gnocchi data source configuration. In order to avoid having issues related to expiring authentication tokens, the default Keystone `token-expiration` (3600 s) parameter was changed to one month (2628000 s):

```
root@namenode:¬# juju config keystone token-expiration=2628000
root@namenode:¬# juju config keystone
...
  token-expiration:
    default: 3600
    description: Amount of time (in seconds) a token should remain valid.
    source: user
    type: int
    value: 2.628e+06
...
```

```
root@namenode:¬# openstack token issue
+------------+----------------------------------------------------------------------+
| Field      | Value                                                                |
+------------+----------------------------------------------------------------------+
| expires    | 2021-06-23T01:51:22+0000                                             |
| id         | gAAAAABgqnn6SkcwKpsGhVa79RqhNr4jwf8je4r9X3LwgLEApA_98m4KqXp |
| project_id | 3d10367c76574f36a007ce9c90761f50                                     |
| user_id    | afd3ffcbb426446296ab30c898581355                                     |
+------------+----------------------------------------------------------------------+
```

4 Fetching Stock Market Data

In order to fetch stock market data a Python3 script was written using the Yahoo! Finance[4] API:

Listing 1: fetch_stock_prices.py

```
1  #!/usr/bin/env python
2
3  import json
4  import datetime
5  import requests
6  import functools
7  import yfinance as yf
8  import urllib.request
9  import dateutil.parser
10 from pytz import timezone
11
12 keystone_token = "gAAAAABgo3ljkfH41aeXCgwj1Ois7rLwtji13hI3MbAXpJ..."
13
14 tickers = ["ENEL.MI", "ISP.MI", "STLA", "ENI.MI", "RACE.MI", ...];
15 tickers_metrics = ["faafee03-ec51-4929-b530-0452eef75464", ...]
16
17 while True:
18   for i in range(len(tickers)):
19     msft = yf.Ticker(tickers[i])
20     print("Processing: " + msft.info['shortName'])
21     pricesDF = msft.history(period="1d", interval="1m")
22
23     # yfinance api timeouts might result in empty dataframes
```

[4] https://finance.yahoo.com/

```
    if not pricesDF.empty:
      priceDateString = str(pricesDF.index[-1])
      priceValue = pricesDF['High'][-1]
      priceDate = dateutil.parser.isoparse(priceDateString)
      priceDate = priceDate.astimezone(timezone('UTC'))
      print(priceDate.strftime("%Y-%m-%dT%H:%M:%S") + " - " + ...)

      # push data to gnocchi
      conditionsSetURL = "http://252.3.238.176:8041/v1/metric/" + ...
      newConditions = [{"timestamp": priceDate.strftime( ...
      params = json.dumps(newConditions).encode('utf8')
      req = urllib.request.Request(conditionsSetURL, data=params, ...
      response = urllib.request.urlopen(req)
      print(response.read().decode('utf8'))
```

The full commented source code of this listing can be found in the GitHub repository under `vallati/python/fetch_stock_prices.py`. For each of the stock market tickers defined, also the corresponding Gnocchi metric ID is defined. For each stock, the highest price of the current trading session is fetched and pushed to Gnocchi. All timestamps are converted to UTC timezone, as expected by Gnocchi.

4.1 Docker Container

A custom Docker container was deployed on the OpenStack controller vm in order to keep the Python script going:

```
# start from an official ubuntu image
FROM ubuntu

# specify the command to be run inside the container at installation
RUN apt-get update && apt-get install -y python3 python3-pip
RUN python3 -m pip install --no-cache-dir --upgrade pip && \
    python3 -m pip install --no-cache-dir gnocchiclient yfinance pandas

# add some file to the image from the host
ADD ./fetch_stock_prices.py /root/

# at startup, run the following command
CMD ["/usr/bin/python3", "/root/fetch_stock_prices.py"]
```

5 Grafana Dashboard

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Etiam aliquam justo eget nunc condimentum, eget iaculis justo venenatis. Aenean id tellus at velit vestibulum tempor nec eu mi. Maecenas lobortis eu mi quis condimentum. Maecenas mi ipsum, semper at felis in, consequat lobortis lorem. Ut imperdiet, ante mattis interdum tristique, orci leo feugiat lorem, facilisis volutpat diam tortor ac quam. Integer faucibus, odio sed consequat sagittis, nunc mi aliquet turpis, ut laoreet velit dolor non nisi. Vivamus vitae libero a est feugiat iaculis ac id mauris.

5.1 Stock Market data Statistics

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Etiam aliquam justo eget nunc condimentum, eget iaculis justo venenatis. Aenean id tellus at velit vestibulum tempor nec eu mi. Maecenas lobortis eu mi quis condimentum. Maecenas mi ipsum, semper at

felis in, consequat lobortis lorem. Ut imperdiet, ante mattis interdum tristique, orci leo feugiat lorem, facilisis volutpat diam tortor ac quam. Integer faucibus, odio sed consequat sagittis, nunc mi aliquet turpis, ut laoreet velit dolor non nisi. Vivamus vitae libero a est feugiat iaculis ac id mauris.

6 Future work

The current deployment can be improved pushing it to the extreme as follows:

- create a new Gnocchi archive policy with a granularity of 1 minute over a timespan of 1 year;
- run a dedicated docker container for each of the stocks we want to monitor to overcome as much as possible Yahoo! Finance API response delays.