CNAF Summer School 2018

Simone Rossi Tisbeni

StoRM metrics and logs parsing and retrieval

Customary workflow for StoRM debugging

A ticket is opened, signaling that an experiment has trouble accessing data. Example files are provided;

The operator search for the example file in Frontend;

Follows it through backend and gridftp to find where it ‘got stuck’;

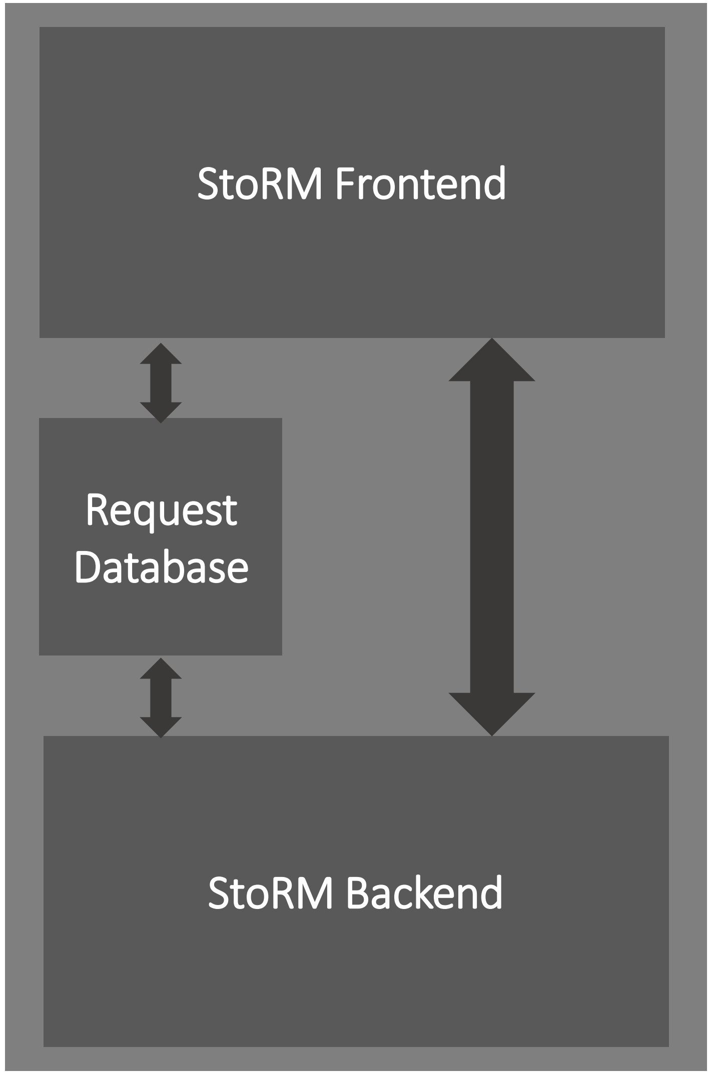
Establish the error timestamp;

Looks to monitoring logs and metrics to determine if StoRM was behaving errouneously.

StoRM
STOrage Resource Management

Disk based storage management service
Built for cluster file system
Direct access (I/O) to shared files and folders

- Frontend – Manages user authentication and stores requests data
- Backend – Executes SRM functionalities, takes care of space and authorization

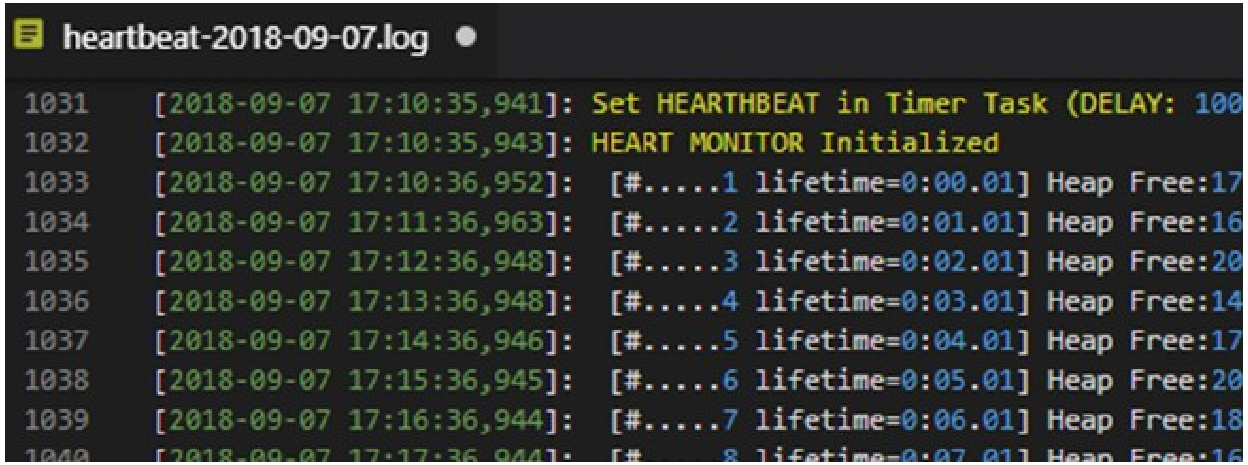


Inserts SRM requests in database
- Logs with requests
- **Logs with operation metrics**

Manages SRM requests
- Logs with success report
- **Logs with operation metrics**

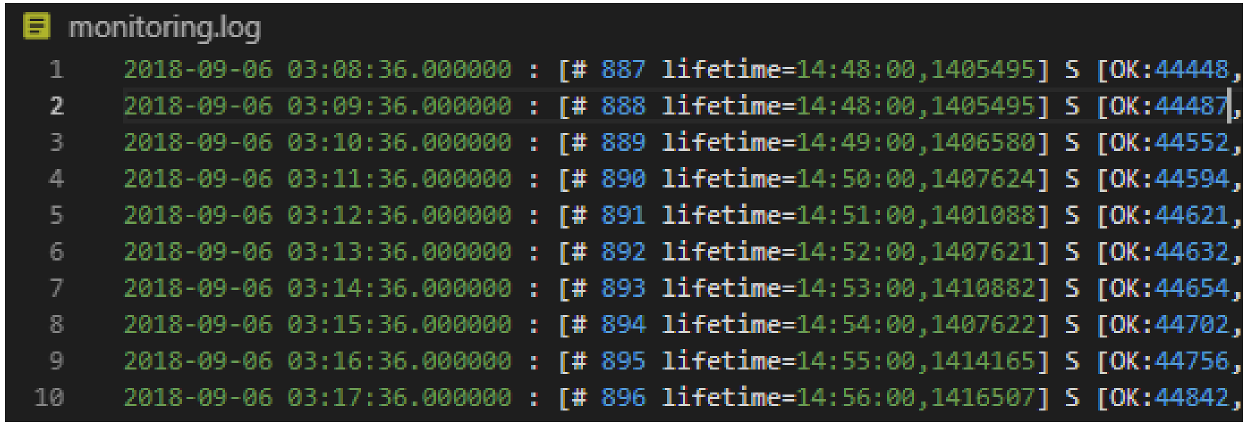
Identification of key components in log file

- Timestamps
- Metrics
- Messages
- Descriptive keys and separators



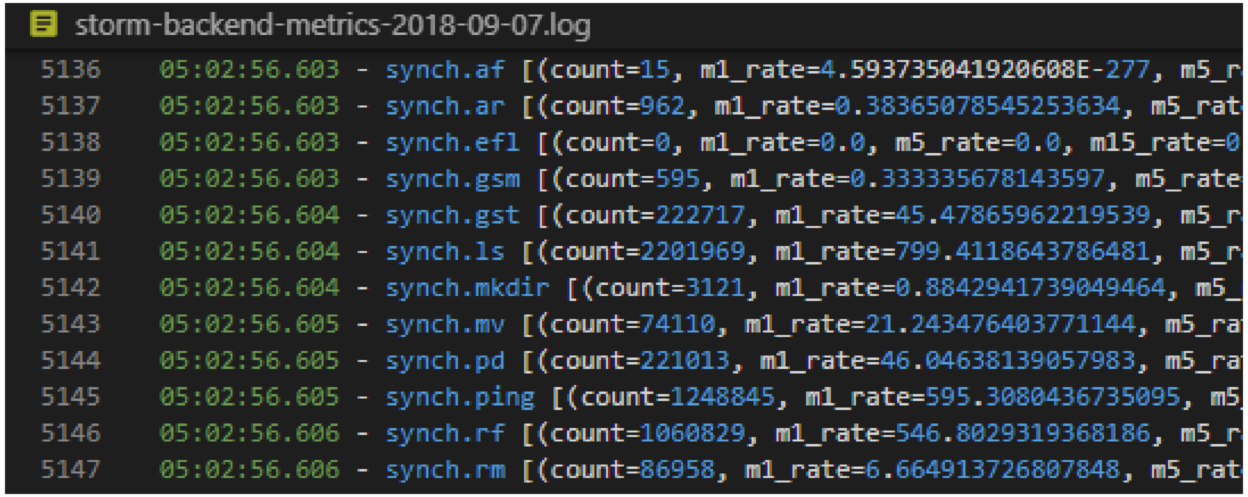
heartbeat-2018-09-07.log ●

```
[2018-09-07 17:10:35,941]: Set HEARTHBEAT in Timer Task (DELAY: 100
[2018-09-07 17:10:35,943]: HEART MONITOR Initialized
[2018-09-07 17:10:36,952]:  [#.....1 lifetime=0:00.01] Heap Free:17
[2018-09-07 17:11:36,963]:  [#.....2 lifetime=0:01.01] Heap Free:16
[2018-09-07 17:12:36,948]:  [#.....3 lifetime=0:02.01] Heap Free:20
[2018-09-07 17:13:36,948]:  [#.....4 lifetime=0:03.01] Heap Free:14
[2018-09-07 17:14:36,946]:  [#.....5 lifetime=0:04.01] Heap Free:17
[2018-09-07 17:15:36,945]:  [#.....6 lifetime=0:05.01] Heap Free:20
[2018-09-07 17:16:36,944]:  [#.....7 lifetime=0:06.01] Heap Free:18
[2018-09-07 17:17:36,944]:  [#.....8 lifetime=0:07.01] Heap Free:16
```



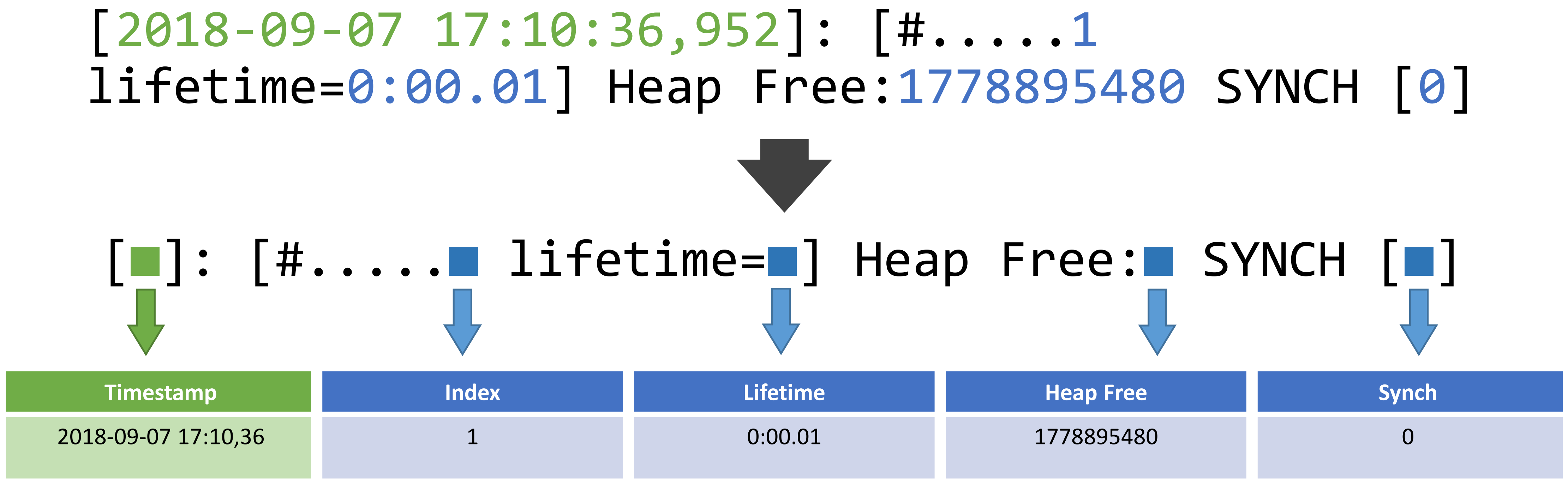
monitoring.log

```
2018-09-06 03:08:36.000000 : [# 887 lifetime=14:48:00,1405495] S [OK:44448,
2018-09-06 03:09:36.000000 : [# 888 lifetime=14:48:00,1405495] S [OK:44487,
2018-09-06 03:10:36.000000 : [# 889 lifetime=14:49:00,1406580] S [OK:44552,
2018-09-06 03:11:36.000000 : [# 890 lifetime=14:50:00,1407624] S [OK:44594,
2018-09-06 03:12:36.000000 : [# 891 lifetime=14:51:00,1401088] S [OK:44621,
2018-09-06 03:13:36.000000 : [# 892 lifetime=14:52:00,1407621] S [OK:44632,
2018-09-06 03:14:36.000000 : [# 893 lifetime=14:53:00,1410882] S [OK:44654,
2018-09-06 03:15:36.000000 : [# 894 lifetime=14:54:00,1407622] S [OK:44702,
2018-09-06 03:16:36.000000 : [# 895 lifetime=14:55:00,1414165] S [OK:44756,
2018-09-06 03:17:36.000000 : [# 896 lifetime=14:56:00,1416507] S [OK:44842,
```



storm-backend-metrics-2018-09-07.log

```
05:02:56.603 - synch.af [(count=15, m1_rate=4.593735041920608E-277, m5_r
05:02:56.603 - synch.ar [(count=962, m1_rate=0.38365078545253634, m5_rat
05:02:56.603 - synch.efl [(count=0, m1_rate=0.0, m5_rate=0.0, m15_rate=0
05:02:56.603 - synch.gsm [(count=595, m1_rate=0.333335678143597, m5_rate
05:02:56.604 - synch.gst [(count=222717, m1_rate=45.47865962219539, m5_r
05:02:56.604 - synch.ls [(count=2201969, m1_rate=799.4118643786481, m5_r
05:02:56.604 - synch.mkdir [(count=3121, m1_rate=0.8842941739049464, m5_
05:02:56.605 - synch.mv [(count=74110, m1_rate=21.243476403771144, m5_ra
05:02:56.605 - synch.pd [(count=221013, m1_rate=46.04638139057983, m5_ra
05:02:56.605 - synch.ping [(count=1248845, m1_rate=595.3080436735095, m5
05:02:56.606 - synch.rf [(count=1060829, m1_rate=546.8029319368186, m5_r
05:02:56.606 - synch.rm [(count=86958, m1_rate=6.664913726807848, m5_rat
```



```
[2018-09-07 17:10:36,952]: [#.....1
lifetime=0:00.01] Heap Free:1778895480 SYNCH [0]
```

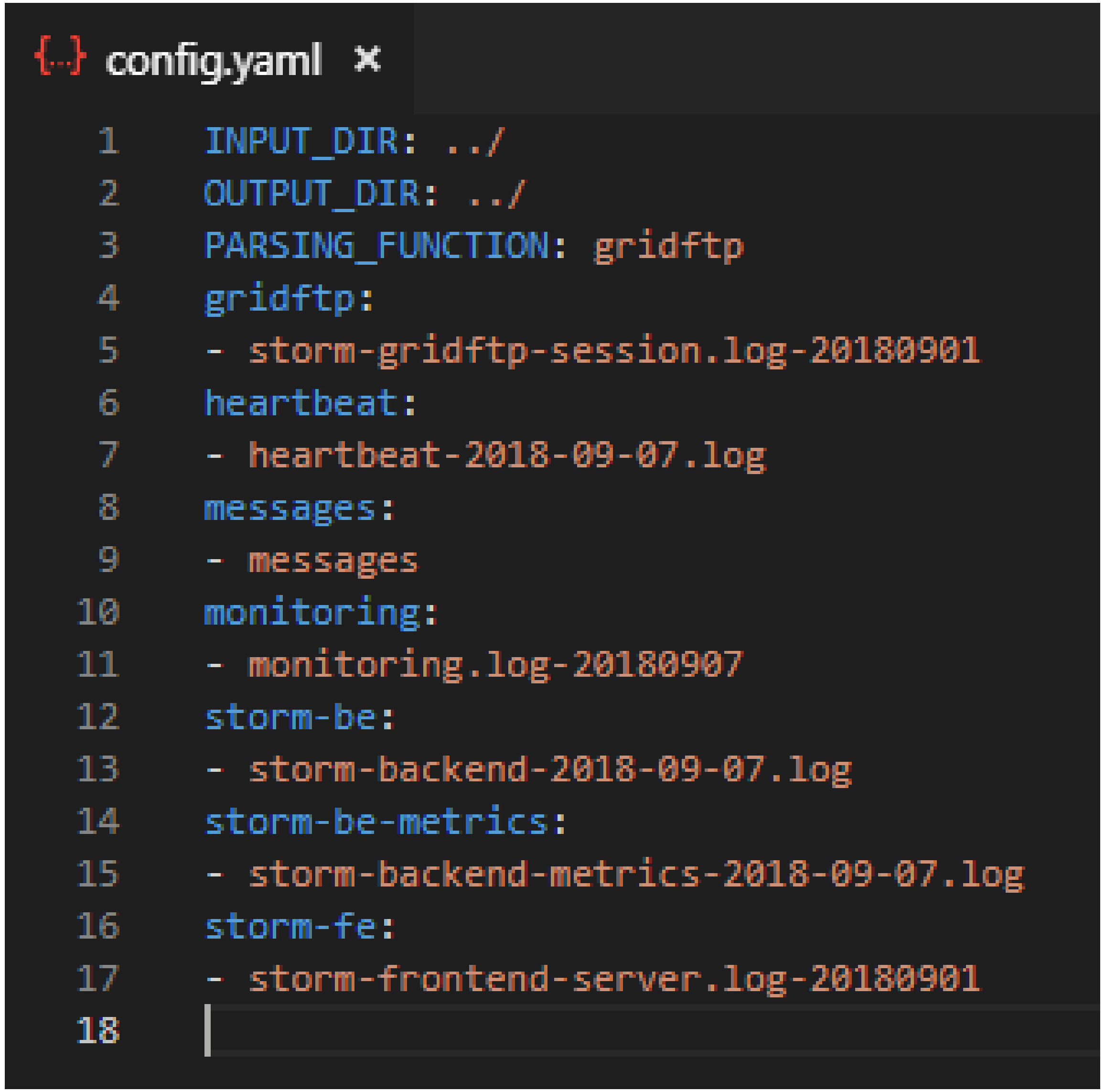
```
[🟩]: [#.....🟦 lifetime=🟦] Heap Free:🟦 SYNCH [🟦]
```

Timestamp	Index	Lifetime	Heap Free	Synch
2018-09-07 17:10,36	1	0:00.01	1778895480	0

Extract information by isolating descriptors

Step by step procedure

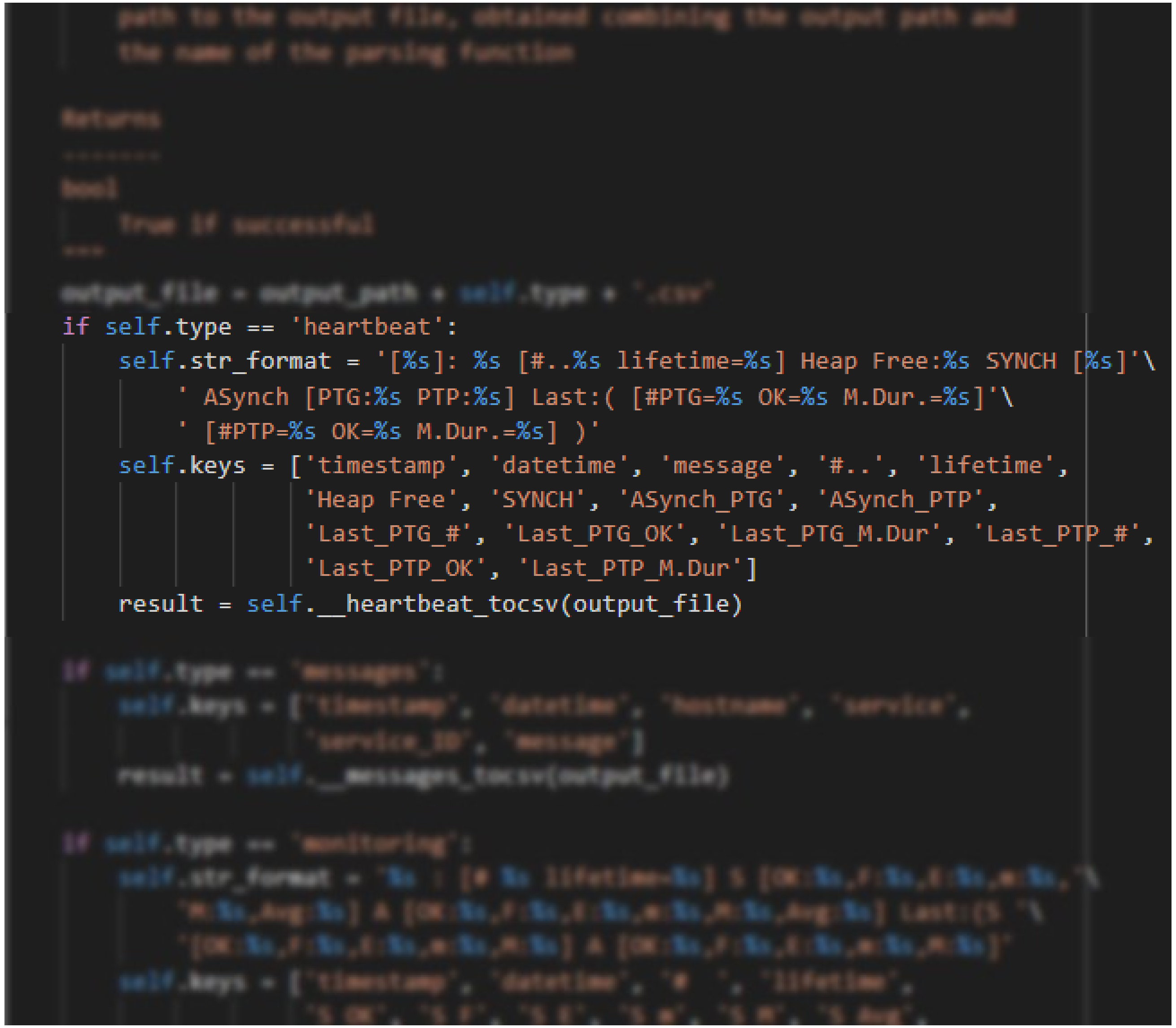
- **Establish working directory and input files**
- Determine format and keys
- Extract the values
- Convert date time in UNIX Epoch Time
- Export the .csv file



```
INPUT_DIR: ../
OUTPUT_DIR: ../
PARSING_FUNCTION: gridftp
gridftp:
- storm-gridftp-session.log-20180901
heartbeat:
- heartbeat-2018-09-07.log
messages:
- messages
monitoring:
- monitoring.log-20180907
storm-be:
- storm-backend-2018-09-07.log
storm-be-metrics:
- storm-backend-metrics-2018-09-07.log
storm-fe:
- storm-frontend-server.log-20180901
```

Step by step procedure

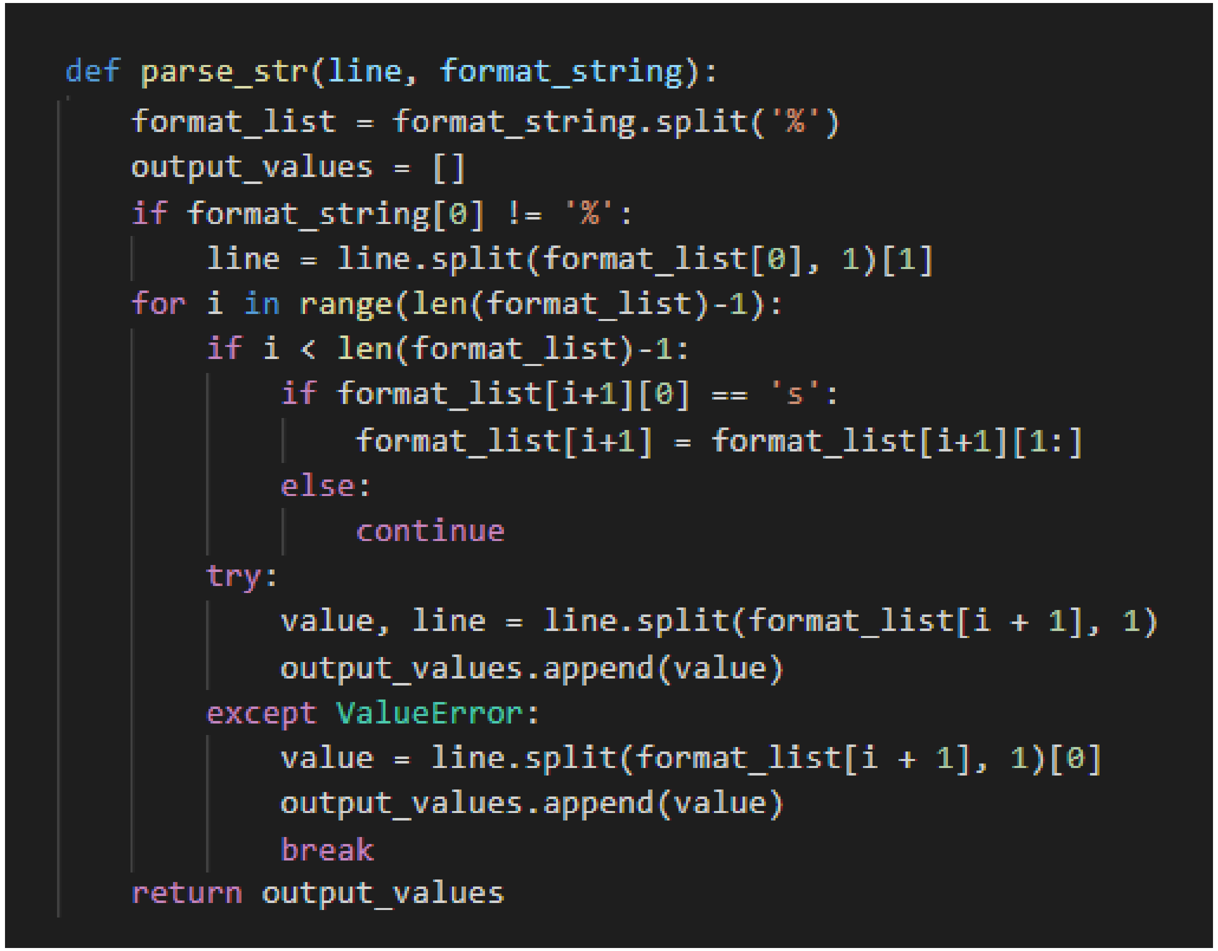
- Establish working directory and input files
- **Determine format and keys**
- Extract the values
- Convert date time in UNIX Epoch Time
- Export the .csv file



```
        path to the output file, obtained combining the output path and
        the name of the parsing function

    Returns
    -------
    bool
        True if successful
    """
    output_file = output_path + self.type + '.csv'
    if self.type == 'heartbeat':
        self.str_format = '[%s]: %s [#..%s lifetime=%s] Heap Free:%s SYNCH [%s]'\
            ' ASynch [PTG:%s PTP:%s] Last:( [#PTG=%s OK=%s M.Dur.=%s]'\
            ' [#PTP=%s OK=%s M.Dur.=%s] )'
        self.keys = ['timestamp', 'datetime', 'message', '#..', 'lifetime',
                     'Heap Free', 'SYNCH', 'ASynch_PTG', 'ASynch_PTP',
                     'Last_PTG_#', 'Last_PTG_OK', 'Last_PTG_M.Dur', 'Last_PTP_#',
                     'Last_PTP_OK', 'Last_PTP_M.Dur']
        result = self.__heartbeat_tocsv(output_file)

    if self.type == 'messages':
        self.keys = ['timestamp', 'datetime', 'hostname', 'service',
                     'service_ID', 'message']
        result = self.__messages_tocsv(output_file)

    if self.type == 'monitoring':
        self.str_format = '%s : [# %s lifetime=%s] S [OK:%s,F:%s,E:%s,m:%s,'\
            'M:%s,Avg:%s] A [OK:%s,F:%s,E:%s,m:%s,M:%s,Avg:%s] Last:(S '\
            '[OK:%s,F:%s,E:%s,m:%s,M:%s] A [OK:%s,F:%s,E:%s,m:%s,M:%s]'
        self.keys = ['timestamp', 'datetime', '#  ', 'lifetime',
                     'S OK', 'S F', 'S E', 'S m', 'S M', 'S Avg',
```

Step by step procedure

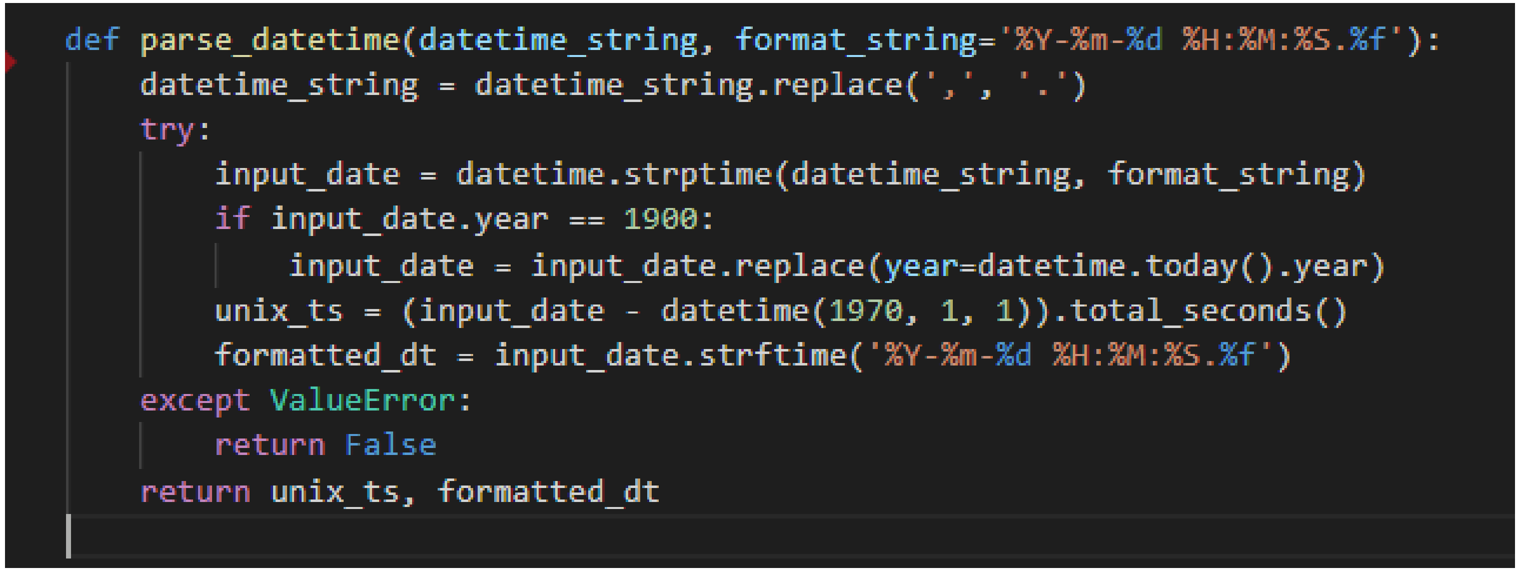
- Establish working directory and input files
- Determine format and keys
- **Extract the values**
- Convert date time in UNIX Epoch Time
- Export the .csv file



```python
def parse_str(line, format_string):
    format_list = format_string.split('%')
    output_values = []
    if format_string[0] != '%':
        line = line.split(format_list[0], 1)[1]
    for i in range(len(format_list)-1):
        if i < len(format_list)-1:
            if format_list[i+1][0] == 's':
                format_list[i+1] = format_list[i+1][1:]
            else:
                continue
        try:
            value, line = line.split(format_list[i + 1], 1)
            output_values.append(value)
        except ValueError:
            value = line.split(format_list[i + 1], 1)[0]
            output_values.append(value)
            break
    return output_values
```

Step by step procedure

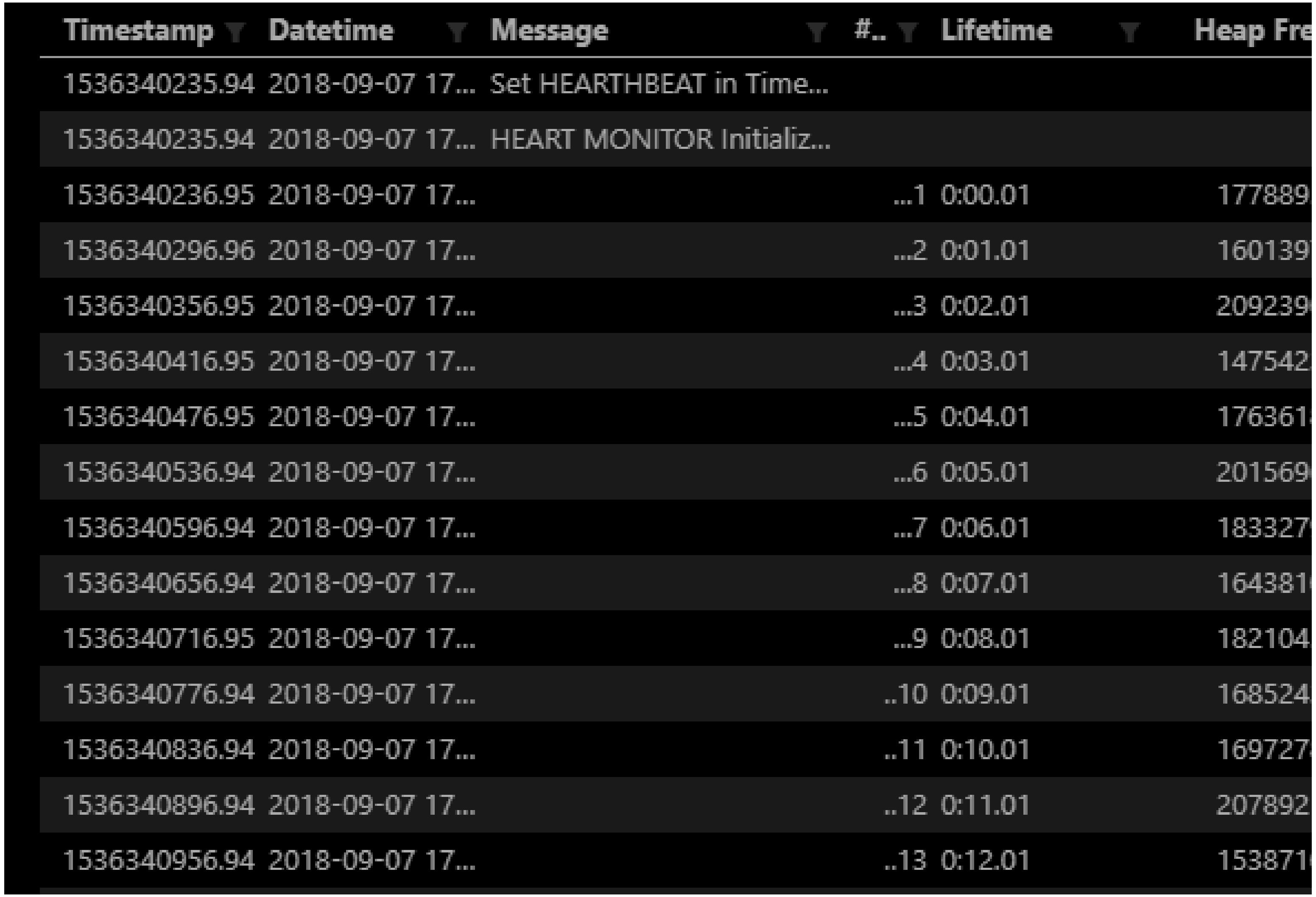
- Establish working directory and input files
- Determine format and keys
- Extract the values
- **Convert date time in UNIX Epoch Time**
- Export the .csv file



```
def parse_datetime(datetime_string, format_string='%Y-%m-%d %H:%M:%S.%f'):
    datetime_string = datetime_string.replace(',', '.')
    try:
        input_date = datetime.strptime(datetime_string, format_string)
        if input_date.year == 1900:
            input_date = input_date.replace(year=datetime.today().year)
        unix_ts = (input_date - datetime(1970, 1, 1)).total_seconds()
        formatted_dt = input_date.strftime('%Y-%m-%d %H:%M:%S.%f')
    except ValueError:
        return False
    return unix_ts, formatted_dt
```

Step by step procedure

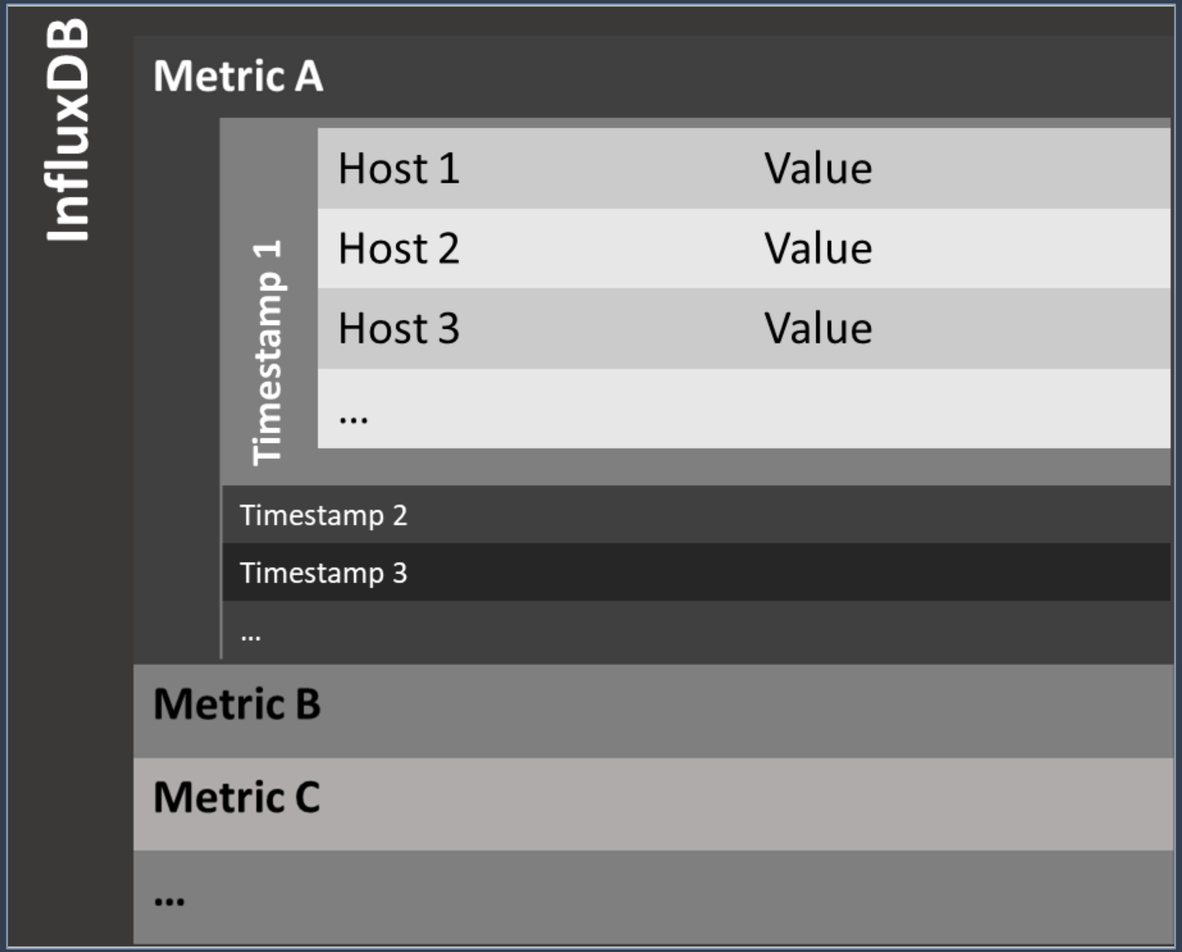
- Establish working directory and input files
- Determine format and keys
- Extract the values
- Convert date time in UNIX Epoch Time
- **Export the .csv file**



Timestamp	Datetime	Message	#..	Lifetime	Heap Fre
1536340235.94	2018-09-07 17...	Set HEARTHBEAT in Time...			
1536340235.94	2018-09-07 17...	HEART MONITOR Initializ...			
1536340236.95	2018-09-07 17...		...1	0:00.01	177889
1536340296.96	2018-09-07 17...		...2	0:01.01	160139
1536340356.95	2018-09-07 17...		...3	0:02.01	209239
1536340416.95	2018-09-07 17...		...4	0:03.01	147542
1536340476.95	2018-09-07 17...		...5	0:04.01	176361
1536340536.94	2018-09-07 17...		...6	0:05.01	201569
1536340596.94	2018-09-07 17...		...7	0:06.01	183327
1536340656.94	2018-09-07 17...		...8	0:07.01	164381
1536340716.95	2018-09-07 17...		...9	0:08.01	182104
1536340776.94	2018-09-07 17...		..10	0:09.01	168524
1536340836.94	2018-09-07 17...		..11	0:10.01	169727
1536340896.94	2018-09-07 17...		..12	0:11.01	207892
1536340956.94	2018-09-07 17...		..13	0:12.01	153871

InfluxDB structure

- Non-relational database
- Optimized for metrics storage
- Different time policies
  - Data is written into 1 week retention policy by default.
  - Every 15 minutes/30 minutes/1 hour the data is down sampled into 1 month/6 month/1 year retention policy.



InfluxDB

Metric A

Timestamp 1

Host 1	Value
Host 2	Value
Host 3	Value
…	

Timestamp 2

Timestamp 3

…

Metric B

Metric C

…

Metrics storage

- Multiple metrics stored for every host

- Measure performance and load of various server depending on the experiment

i.e. ATLAS experiment

- storm-atlas.cr.cnaf.infn.it

  frontend and backend

- storm-fe-atlas-07.cr.cnaf.infn.it

  second frontend

- ds-808.cr.cnaf.infn.it

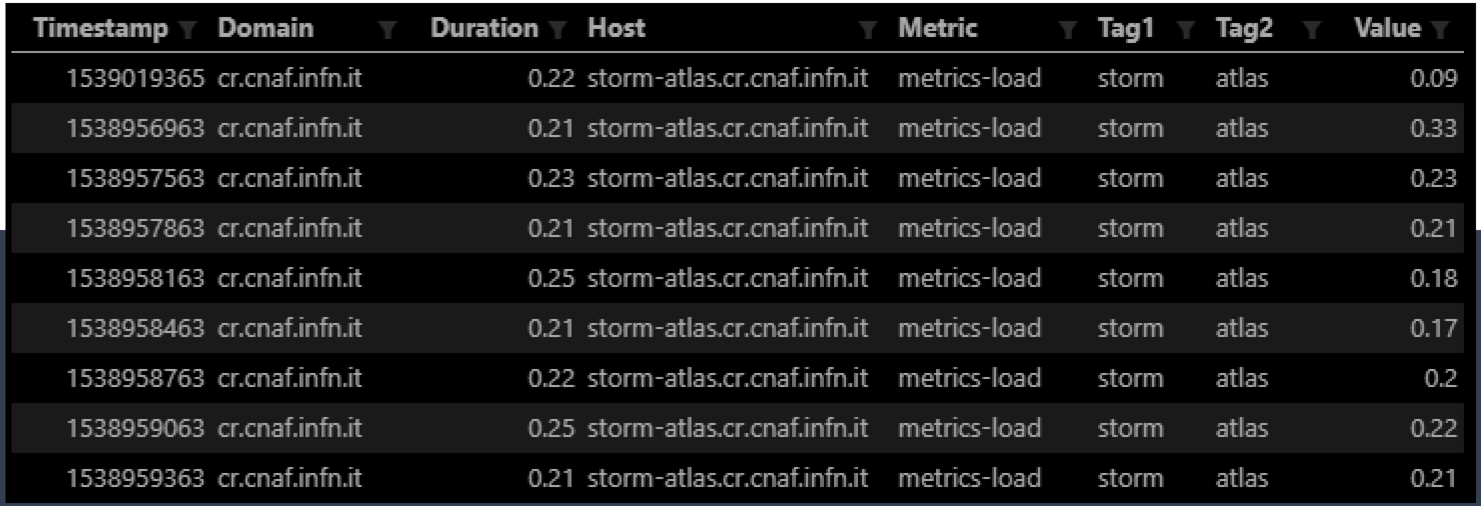
- ds-908.cr.cnaf.infn.it

  gridftp

Querying the database

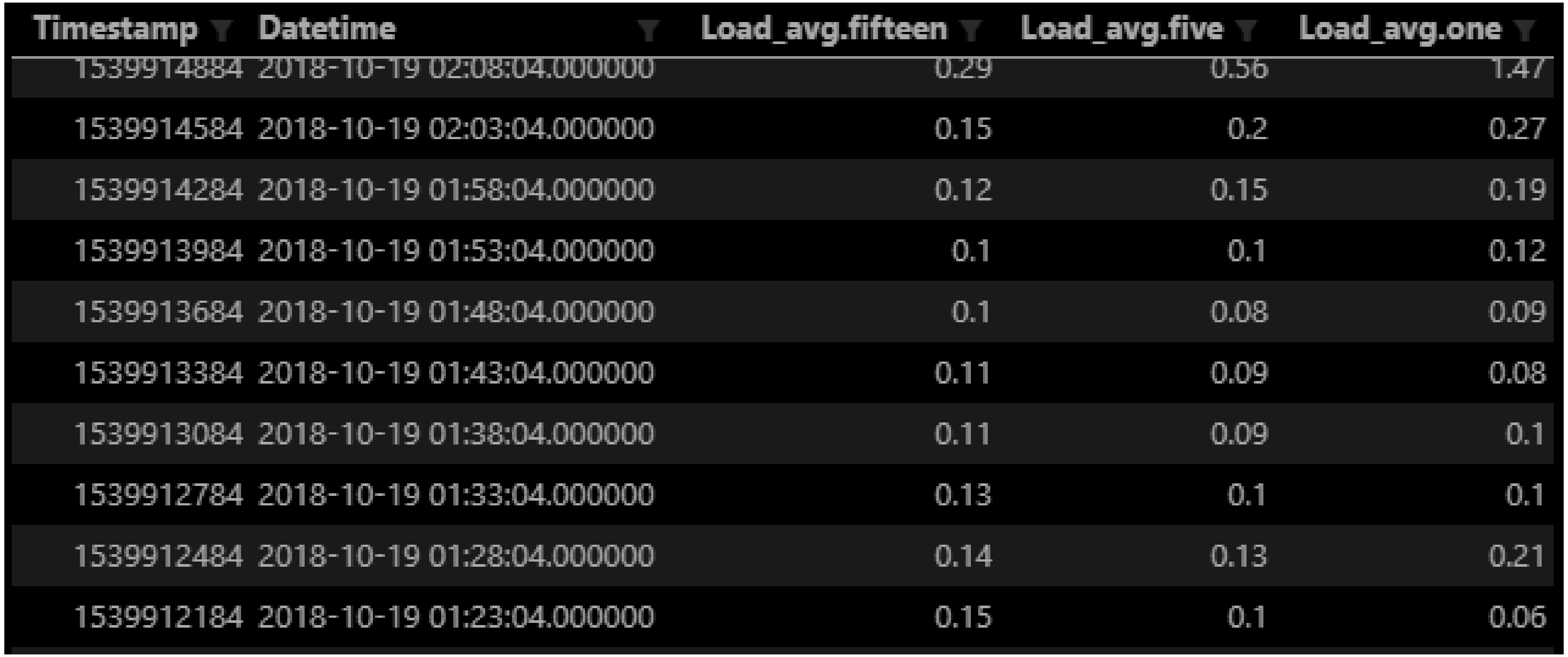
- Establish a connection to the client
- Determine the measurement types
- Query for a specific host

```
influx -host=HOST -port=PORT -username=“NAME" -
password=“PASSWORD" -database=“DATABASE”



SHOW MEASUREMENTS



SELECT * FROM ‘MEASUREMENT’ WHERE "host" =
‘HOSTNAME’
```



Timestamp	Domain	Duration	Host	Metric	Tag1	Tag2	Value
1539019365	cr.cnaf.infn.it	0.22	storm-atlas.cr.cnaf.infn.it	metrics-load	storm	atlas	0.09
1538956963	cr.cnaf.infn.it	0.21	storm-atlas.cr.cnaf.infn.it	metrics-load	storm	atlas	0.33
1538957563	cr.cnaf.infn.it	0.23	storm-atlas.cr.cnaf.infn.it	metrics-load	storm	atlas	0.23
1538957863	cr.cnaf.infn.it	0.21	storm-atlas.cr.cnaf.infn.it	metrics-load	storm	atlas	0.21
1538958163	cr.cnaf.infn.it	0.25	storm-atlas.cr.cnaf.infn.it	metrics-load	storm	atlas	0.18
1538958463	cr.cnaf.infn.it	0.21	storm-atlas.cr.cnaf.infn.it	metrics-load	storm	atlas	0.17
1538958763	cr.cnaf.infn.it	0.22	storm-atlas.cr.cnaf.infn.it	metrics-load	storm	atlas	0.2
1538959063	cr.cnaf.infn.it	0.25	storm-atlas.cr.cnaf.infn.it	metrics-load	storm	atlas	0.22
1538959363	cr.cnaf.infn.it	0.21	storm-atlas.cr.cnaf.infn.it	metrics-load	storm	atlas	0.21

Every query is done through the Python client:
InfluxDB-Python



Timestamp	Datetime	Load_avg.fifteen	Load_avg.five	Load_avg.one
1539914884	2018-10-19 02:08:04.000000	0.29	0.56	1.47
1539914584	2018-10-19 02:03:04.000000	0.15	0.2	0.27
1539914284	2018-10-19 01:58:04.000000	0.12	0.15	0.19
1539913984	2018-10-19 01:53:04.000000	0.1	0.1	0.12
1539913684	2018-10-19 01:48:04.000000	0.1	0.08	0.09
1539913384	2018-10-19 01:43:04.000000	0.11	0.09	0.08
1539913084	2018-10-19 01:38:04.000000	0.11	0.09	0.1
1539912784	2018-10-19 01:33:04.000000	0.13	0.1	0.1
1539912484	2018-10-19 01:28:04.000000	0.14	0.13	0.21
1539912184	2018-10-19 01:23:04.000000	0.15	0.1	0.06

Extract csv tables

- Should follow a host-first structure
- Should merge different measurements by category
- Should maintain the UNIX timestamp format

Log frequency

One line every minute

- Heartbeat log
- Monitorging log

Multiple lines every minute

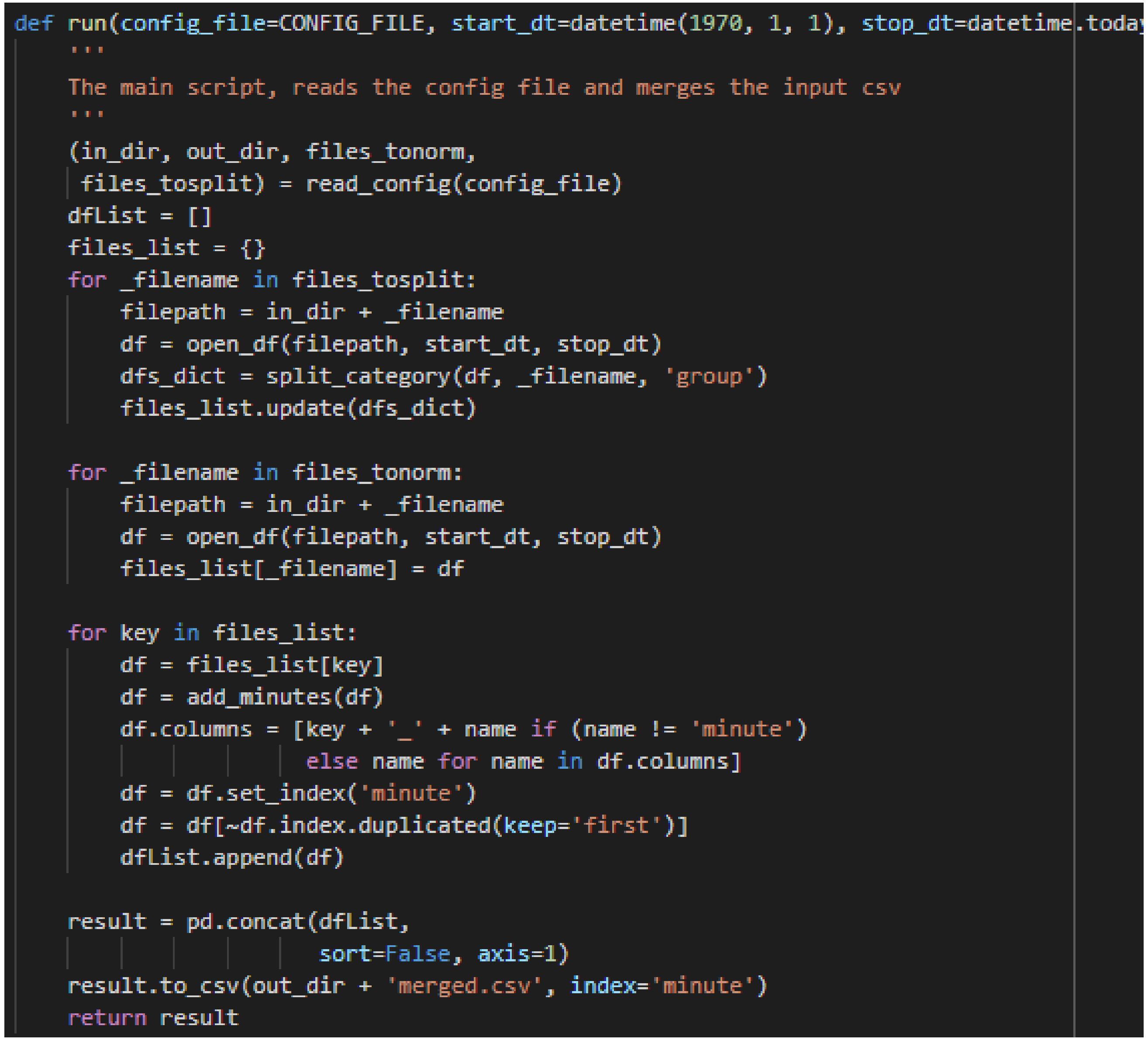
- Backend metrics logs
  (one for each command)

One line every five minutes:

- InfluxDB metrics

Concatenation rules

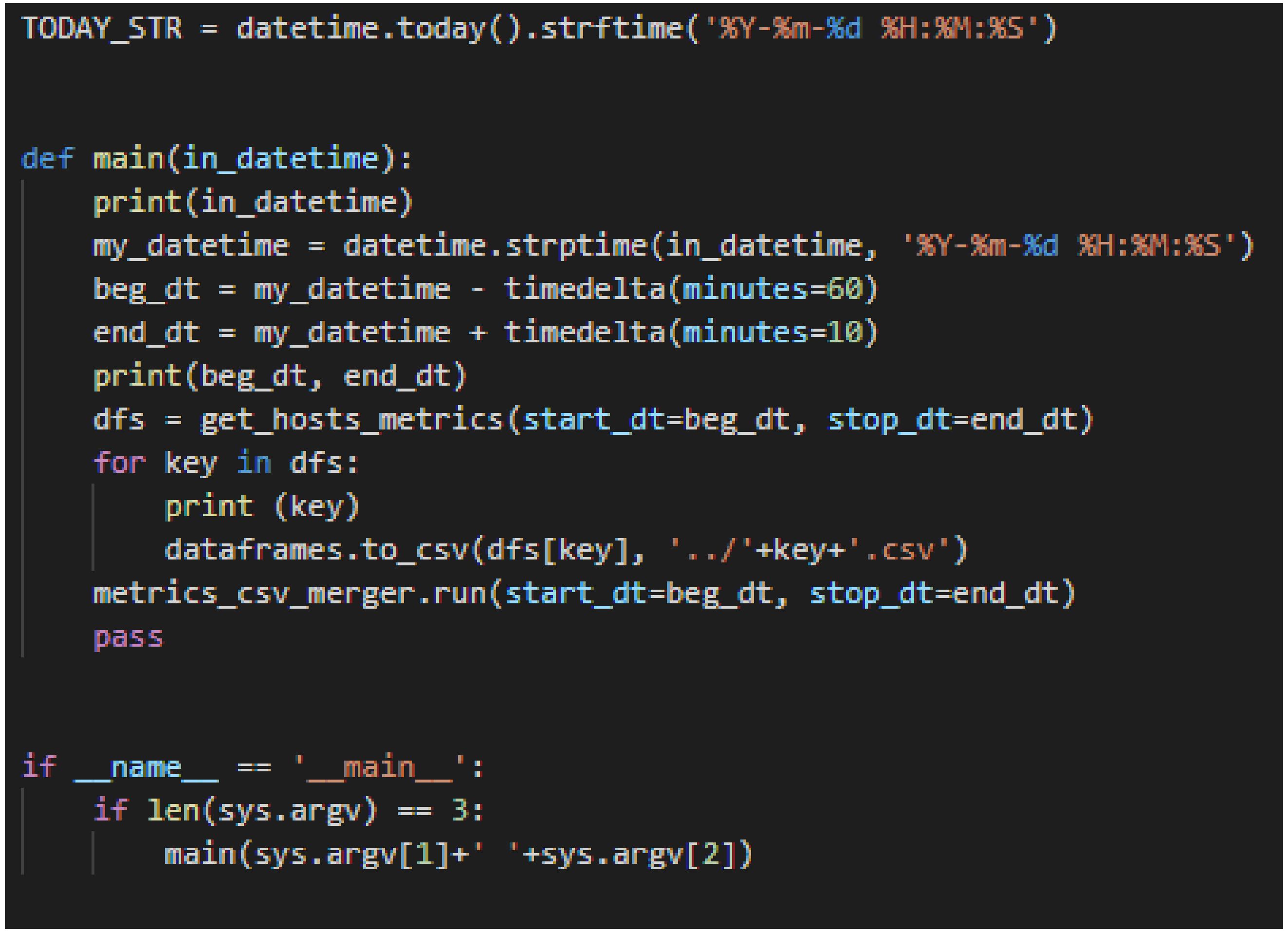
- Backend metrics are split by type
- Timestamp is rounded off to one-minute precision
- In case of overlap the more recent is kept
- Every .csv is concatenated and ordered by timestamp



```
def run(config_file=CONFIG_FILE, start_dt=datetime(1970, 1, 1), stop_dt=datetime.today
    '''
    The main script, reads the config file and merges the input csv
    '''
    (in_dir, out_dir, files_tonorm,
     files_tosplit) = read_config(config_file)
    dfList = []
    files_list = {}
    for _filename in files_tosplit:
        filepath = in_dir + _filename
        df = open_df(filepath, start_dt, stop_dt)
        dfs_dict = split_category(df, _filename, 'group')
        files_list.update(dfs_dict)

    for _filename in files_tonorm:
        filepath = in_dir + _filename
        df = open_df(filepath, start_dt, stop_dt)
        files_list[_filename] = df

    for key in files_list:
        df = files_list[key]
        df = add_minutes(df)
        df.columns = [key + '_' + name if (name != 'minute')
                      else name for name in df.columns]
        df = df.set_index('minute')
        df = df[~df.index.duplicated(keep='first')]
        dfList.append(df)

    result = pd.concat(dfList,
                       sort=False, axis=1)
    result.to_csv(out_dir + 'merged.csv', index='minute')
    return result
```

Query for specific timestamp

- Queries InfluxDB using the most accurate retention policy
- Creates a merged logs database with values included between two timestamp



```
TODAY_STR = datetime.today().strftime('%Y-%m-%d %H:%M:%S')


def main(in_datetime):
    print(in_datetime)
    my_datetime = datetime.strptime(in_datetime, '%Y-%m-%d %H:%M:%S')
    beg_dt = my_datetime - timedelta(minutes=60)
    end_dt = my_datetime + timedelta(minutes=10)
    print(beg_dt, end_dt)
    dfs = get_hosts_metrics(start_dt=beg_dt, stop_dt=end_dt)
    for key in dfs:
        print (key)
        dataframes.to_csv(dfs[key], '../'+key+'.csv')
    metrics_csv_merger.run(start_dt=beg_dt, stop_dt=end_dt)
    pass


if __name__ == '__main__':
    if len(sys.argv) == 3:
        main(sys.argv[1]+' '+sys.argv[2])
```

Conclusions

- Increased the readability of logs
- Introduced toolkit to access specific logs and metrics
- Prepared the foundation for machine learning algorithms
- The code produced is well documented to allow for easier pick-up by different people

For more information on InfluxDB

Documentation: https://docs.influxdata.com/influxdb/v1.7/

Python library: https://influxdb-python.readthedocs.io/en/latest/

The repository for the code presented is available through the following git:
https://baltig.infn.it/summerstudentscnaf/log-parsing

Simone Rossi Tisbeni

Grad student in Applied Physics

A.A 2018/19

Will continue working on predictive maintenance at CNAF for his graduate thesis with Prof. Bonacorsi Daniele and Martelli Barbara