HTCondor Monitoring with ELK @ BNL

ElasticSearch Workshop, Sep 30th 2019, FNAL

Jose Caballero <jcaballero@bnl.gov>

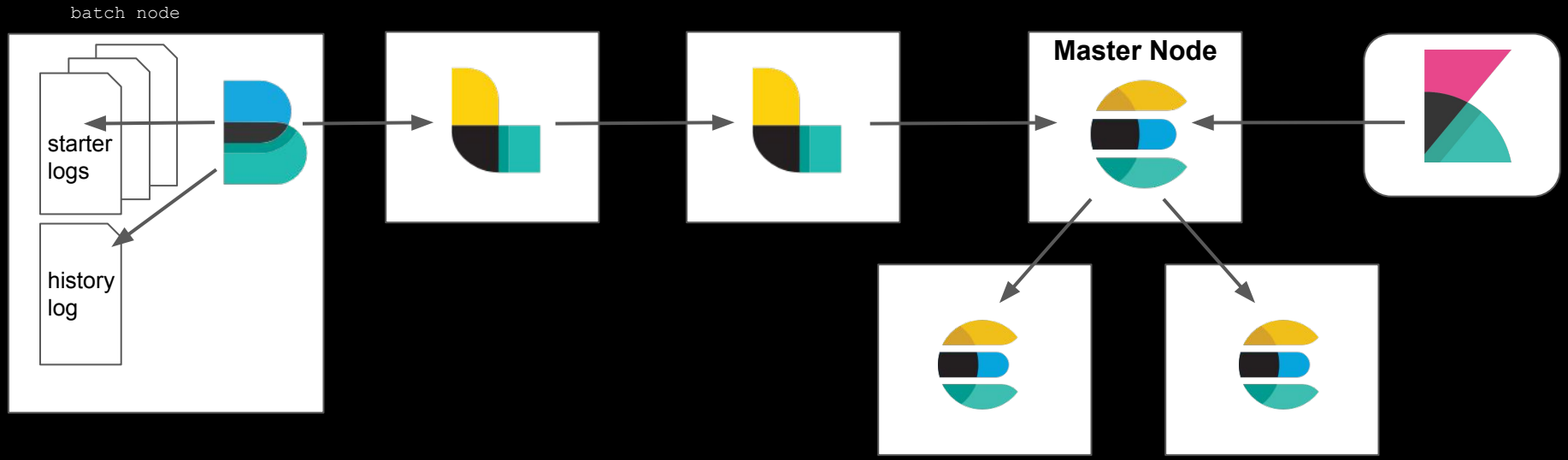
Intro

Different indices are stored in the ElasticSearch cluster @ BNL.

In this talk I will focus on the LogStash pipeline for HTCondor clusters monitoring:

- The implementation is somehow unique.
- The rest is pretty standard.

High level architecture



Goal

1. Collect several pieces of information for each job as soon as they start.
    - StarterLog.slot<XYZ>
2. Complete job information once the job finished.
    - StarterLog.slot<XYZ>
    - startd_history

The challenge

This is not quite a standard LogStash pipeline:

- We need to consider several lines at once.
  - But they are not consecutive.
  - No common tag to identify the ones for the same job.
- We only send data to the output when we reach:
  - the line containing username, or
  - the line for end of job.
- We need to spot jobs that crashed (line of end of job is missing) and send proper data to the output. Otherwise, they would stay forever in ElasticSearch as running.
- All of the above for each StarterLog file, from many batch nodes. We read many of them at the same time, and the LogStash pipeline gets their lines interleaved.

The existing LogStash plugins are not good enough (to my current knowledge).

Write almost the entire algorithm within the Ruby plugin.

The challenge (ii): mixing StarterLog data with condor_history data

Once jobs are done, we complete their representation in ElasticSearch with data from the condor_history file:

- We mix data coming from StarterLog files with data from startd_history file.
  - This seems to be quite innovative in our community. No one has done it (?)
- To update an existing document in ElasticSearch, it is referred by 2 variables:
    - its index
    - its unique ID
  - We construct the index from the day the job started, and the unique ID is based on the classAd GlobalJobId: both are available in the StarterLog and the history log.
- As two separate streams of data are mixed in LogStash, we need to fix the filter plugins to know which type of info is being processed at any time.

Example: the Starterlog files for starting jobs

```
 9 02/15/19 17:20:06.613 **********************************************************
10 02/15/19 17:20:06.613 ** condor_starter (CONDOR_STARTER) STARTING UP
11 02/15/19 17:20:06.613 ** /usr/sbin/condor_starter
12 02/15/19 17:20:06.613 ** SubsystemInfo: name=STARTER type=STARTER(8) class=DAEMON(1)
13 02/15/19 17:20:06.613 ** Configuration: subsystem:STARTER local:<NONE> class:DAEMON
14 02/15/19 17:20:06.613 ** $CondorVersion: 8.8.0 Jan 04 2019 BuildID: racf $
15 02/15/19 17:20:06.613 ** $CondorPlatform: X86_64-ScientificLinux_7.6 $
16 02/15/19 17:20:06.613 ** PID = 13304
17 02/15/19 17:20:06.613 ** Log last touched time unavailable (No such file or directory)
18 02/15/19 17:20:06.613 **********************************************************


34 02/15/19 17:20:06.637 Shadow version: $CondorVersion: 8.8.0 Jan 04 2019 BuildID: racf $
35 02/15/19 17:20:06.637 Submitting machine is "rcas6008.rcf.bnl.gov"
36 02/15/19 17:20:06.638 Instantiating a StarterHookMgr


207 02/15/19 17:20:06.811 Job 13088420.28 set to execute immediately
208 02/15/19 17:20:06.811 Starting a VANILLA universe job with ID: 13088420.28
209 02/15/19 17:20:06.811 In OsProc::OsProc()


231 02/15/19 17:20:06.820 ENFORCE_CPU_AFFINITY not true, not setting affinity
232 02/15/19 17:20:06.820 Running job as user tchuang
233 02/15/19 17:20:06.838 track_family_via_cgroup: Tracking PID 13484 via cgroup htcondor/condor_home_condor_local_sdcc_execute_                gov.
```

start time

submit host

job id

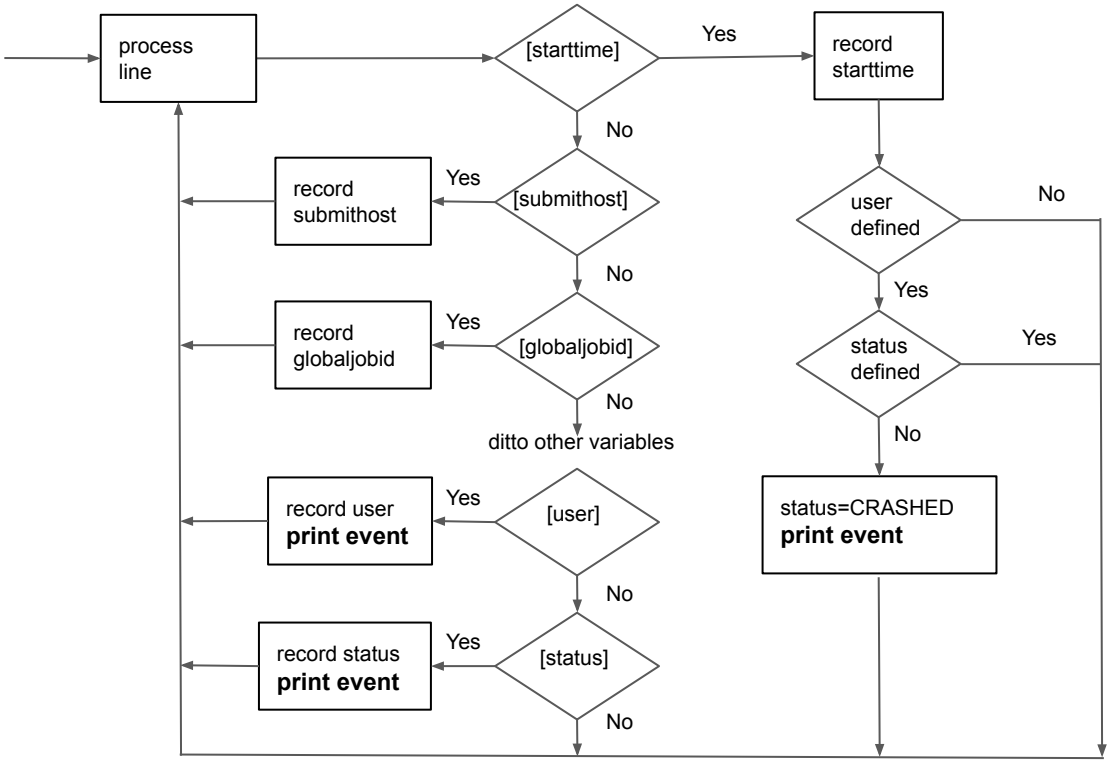
username

Output

The first piece of data sent to ElasticSearch, upon job start, looks more or less like this:

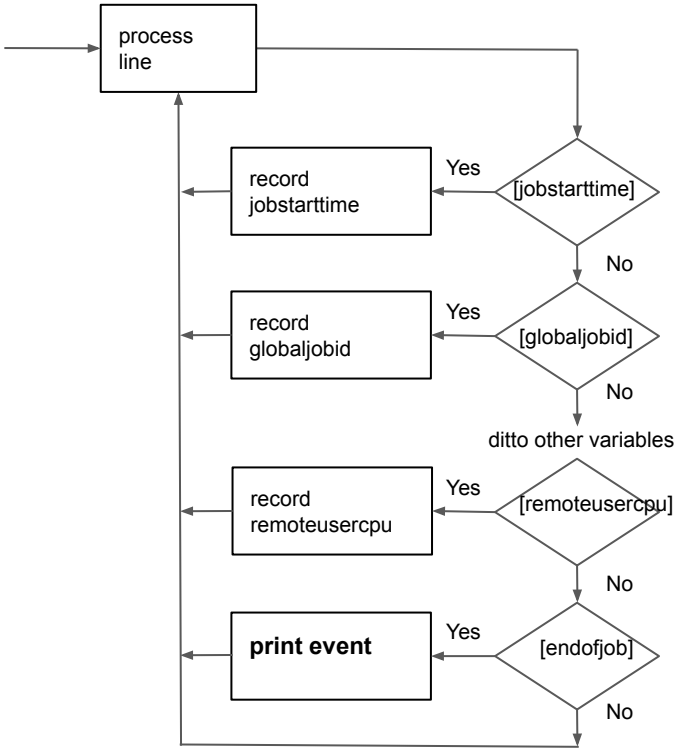
```
{
          "logfile" => "StarterLog.slot1_3",
       "submithost" => "myhost.bnl.gov",
             "user" => "johndoe",
        "starttime" => "04/08/19 02:27:53.345",
     "globaljobid" => "myhost.bnl.gov#8885009.48#1554704830",
            "jobid" => "8885009.48",
   "executionhost" => "anode1234.bnl.gov",
            "index" => "htcondorjobs-04.08.19",
       "@timestamp" => 2019-04-08T06:27:53.345Z
}
```

Implementation: algorithm for each job (StarterLog.slotXYZ)



Implemented in the LogStash's filter plugin Ruby.

Implementation: algorithm for each job (startd_history)



Implemented in the LogStash's filter plugin Ruby.

Implementation: separate algorithms depending on the type of log file

```
filebeat.inputs:

- type: log
  paths:
    - /var/log/condor/sdcc/startd_history
  fields: {log_type: history}

- type: log
  paths:
    - /var/log/condor/sdcc/StarterLog.slot1_*
  fields: {log_type: starterlog}

output.logstash:
  hosts: ["localhost:5044"]
```

```
filter {

   if [fields][log_type] == "starterlog" {

        # here the code to parse StarterLog data

   } else if [fields][log_type] == "history" {

        # here the code to parse startd_history data

   }

}
```

Implementation: distinguish content by file/host

```
filter{

    mutate {
            split => ["source", "/"]
            add_field => { "logfile" => "%{[source][-1]}"  }
            add_field => { "origin" => "%{logfile}@%{[beat][hostname]}"  }
    }


    ruby {
        init => '
            @starttime_h = Hash.new
            @globaljobid_h = Hash.new
            @trial_h = Hash.new
            @jobid_h = Hash.new
            @submithost_h = Hash.new
            @user_h = Hash.new
            ...
        '

        code => '

            if event.get("starttime")
                @starttime_h[@origin] = event.get("starttime")
            ...
        '


    }
}
```

Every piece of data is stored in a Hash. The key of the hash is a combination:

input file × host

Implementation: job's starttime as @timestamp in ES

```
filter{

   date {
        match => [ "starttime" , "MM/dd/yy HH:mm:ss.SSS", "MM/dd/yy HH:mm:ss", "MM/dd/yy HH:mm:ss.1000" ]
    }

}
```

Use the job start time as timestamp, instead of the day and time the data is sent to ElasticSearch.

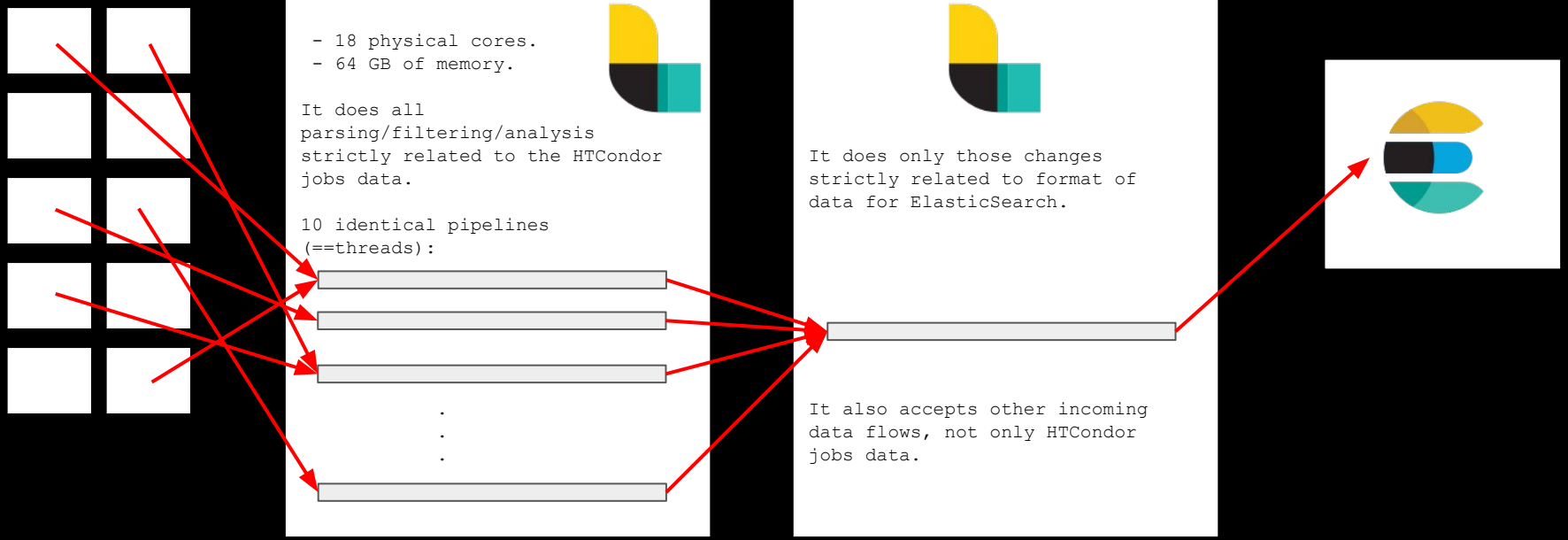
We send info several times for each job. In particular, after completion, which could be several days after job started.

It facilitates searching jobs in Kibana if the timestamp is always the time the job started.

Implementation: job's starttime and globaljobid for the index/doc_id in ES

```
filter{

   ruby {
      code => '
           date = Date.strptime(event.get("starttime"), "%m/%d/%y %H:%M:%S").strftime("%Y.%m.%d")
           event.set("index", "htcondorjobs-" + date)

           event.set("document_id", event.get("globaljobid") + "_" + event.get("trial").to_s)
      '
   }

}


output {
    elasticsearch {
        ...
        index => "%{index}"
        document_id => "%{document_id}"
        ...
    }
}
```

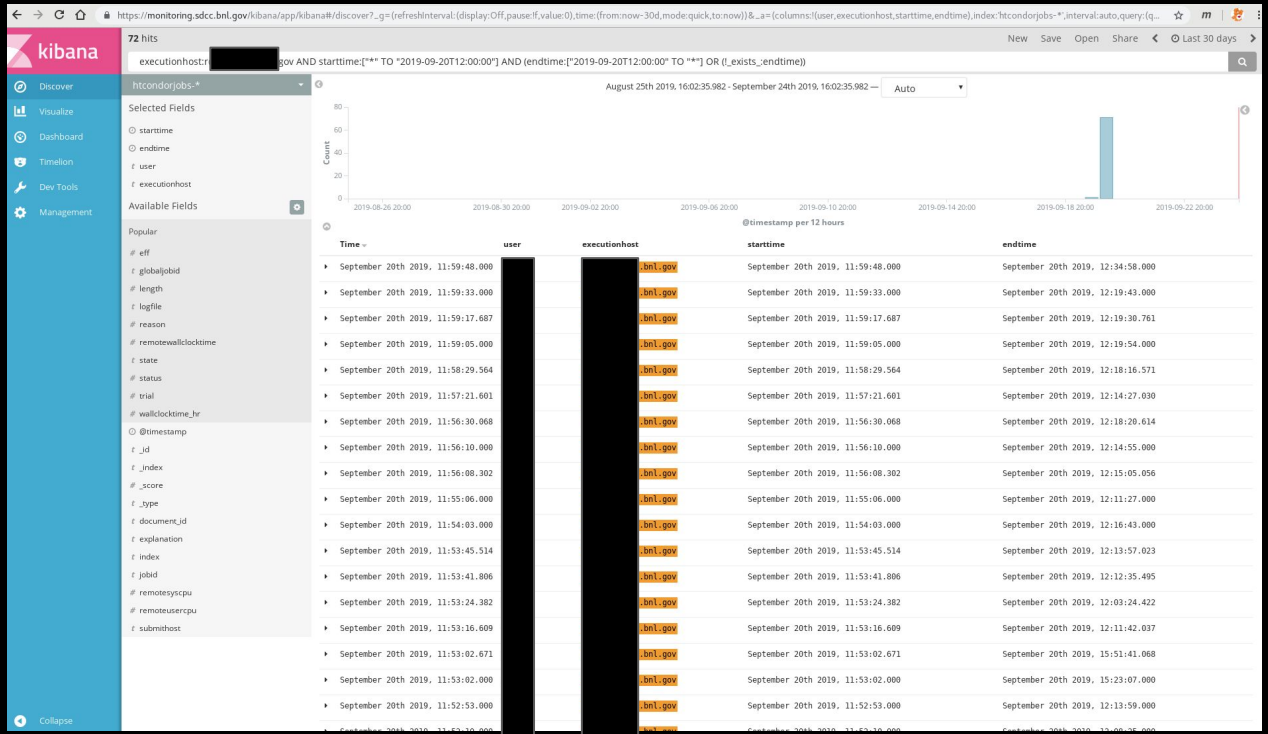
LogStash high level architecture



Example of usage: a very common query (other visualizations in the backup slides)

One of the most common needs: who was running on a particular host at a given time?

```
executionhost:<MyHost>
AND
starttime:["*" TO "2019-09-20T12:00:00"]
AND
(
  endtime:["2019-09-20T12:00:00" TO "*"]
  OR
  ( !_exists_:endtime)
)
```



Backup slides

List of attributes recorded in ElasticSearch

From FileBeat	From StarterLog	From history log	Calculated/harcoded in LogStash
executionhost	starttime	starttime	explanation
logfile	endtime	globaljobid	state
	status	remotesyscpu	eff
	reason	remoteusercpu	
	user	remotewallclocktime	
	submithost		
	trial		
	jobid		
	globaljobid		

$$\frac{RemoteUserCpu + RemoteSysCpu}{RemoteWallClockTime * CpusProvisioned}$$

Implementation (extra): add the execution host

```
filter {


        mutate {
             add_field => { "executionhost" => "%{[host][name]}" }
        }


}
```

Implementation (extra): add the input logfile

```
filter {

       mutate {
            split => ["source", "/"]
            add_field => { "logfile" => "%{[source][-1]}"  }
        }

}
```

Knowing the exact log file is very helpful for troubleshooting.

Implementation (extra): remove unnecessary data

```
filter{

       prune {
            whitelist_names => ["globaljobid",
                                "jobid",
                                "user",
                                "submithost",
                                "starttime",
                                "endtime",
                                "status",
                                "reason",
                                "executionhost",
                                "logfile",
                                "explanation"]
        }

}
```

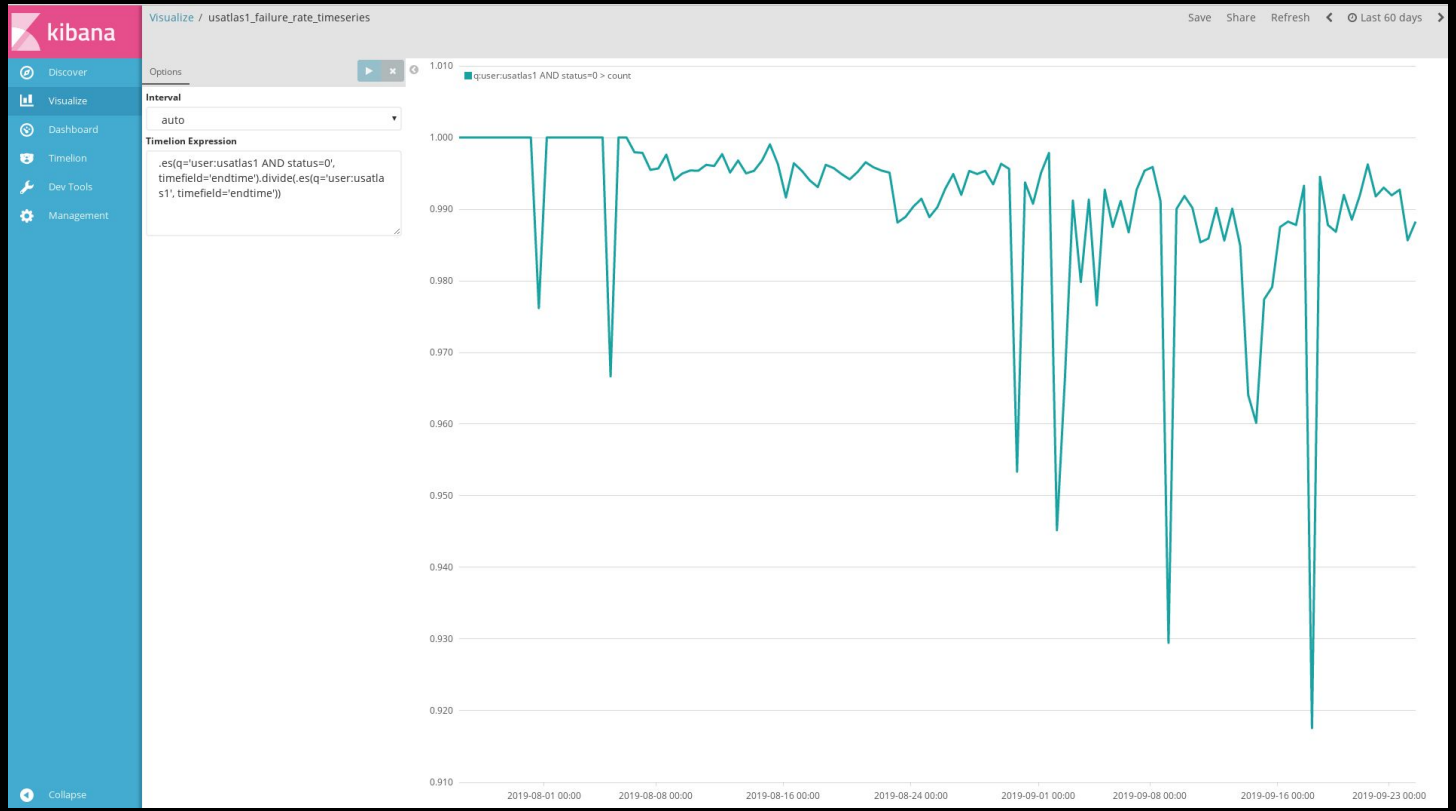
Idea is not to dump too much useless data into ElasticSearch.

Implementation (extra): filter too old data upon start

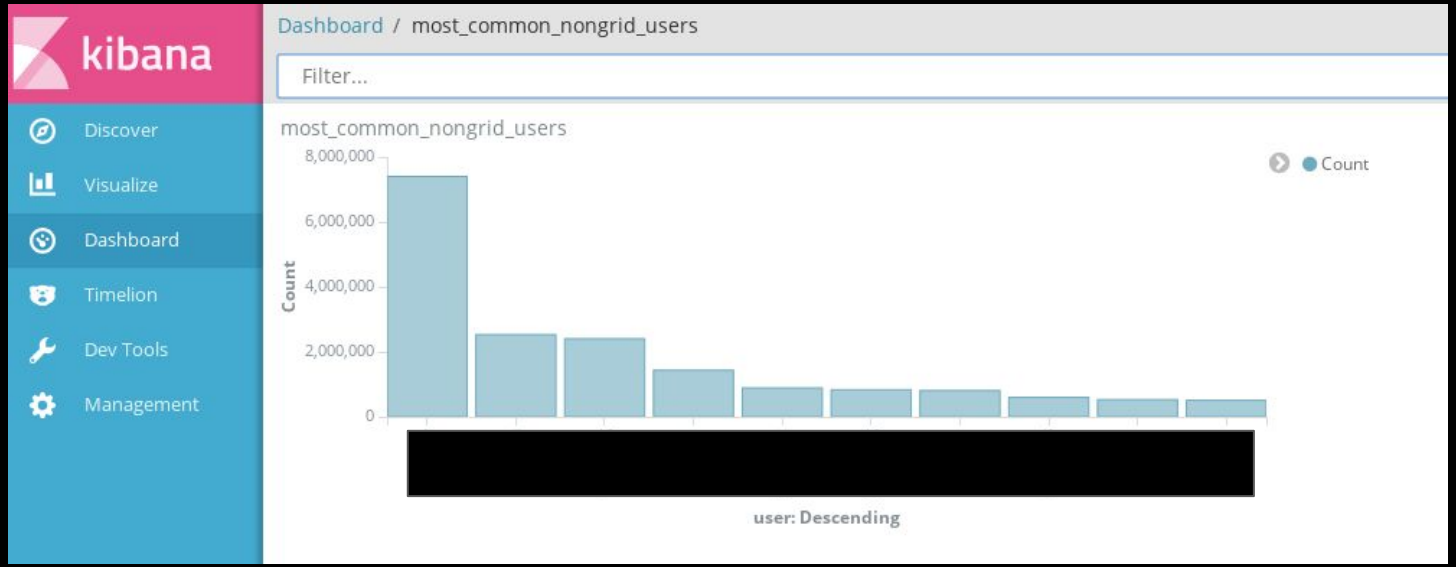
```
filter{

    date {
        match => ["eventtimestamp", "MM/dd/yy HH:mm:ss.SSS"]
        target => "eventtimestamp"
    }


    ruby {

        if (! $recentenough_h[$source] ) && event.get("starttime")
            if (Time.now.to_i - event.get("eventtimestamp").to_i) < 3600*24*10
                $recentenough_h[$source] = true
            end
        end

        if ! $recentenough_h[$source]
            event.cancel


        elsif
            # rest of the logic here

    }
}
```

StarterLog files have data even 1 year old. To avoid parsing it, and dumping it into ES (too much space) at the very beginning, we filter based on the timestamp.

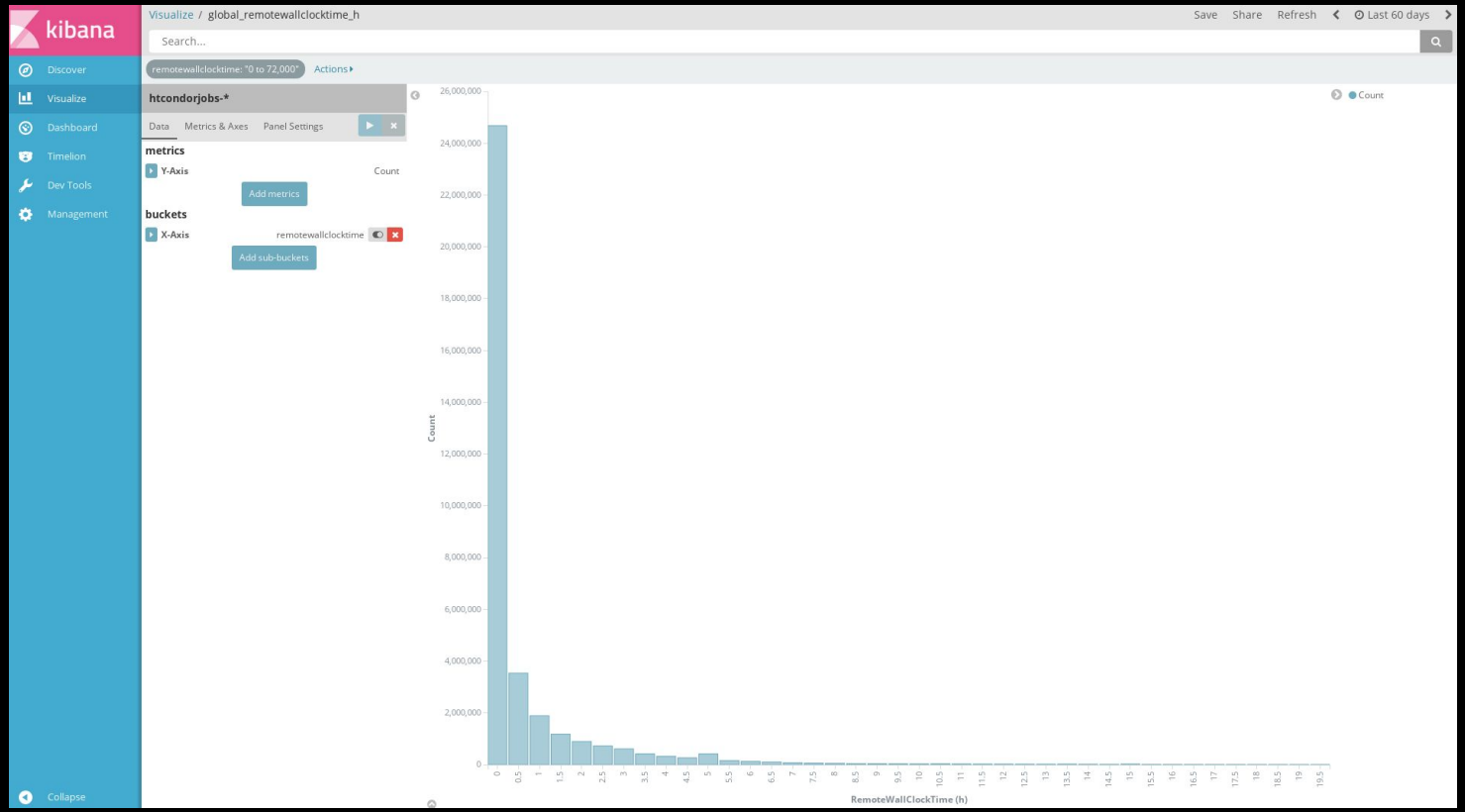
Some basic visualizations: timeseries of ATLAS failure rate with Timelion



Some basic visualizations: the 10 most active local users



Some basic visualizations: histogram for ClassAd RemoteWallClockTime



Some basic visualizations: timeseries for RemoteWallClockTime for ATLAS jobs