OMNI/PRESCRIBE MONITORING USAGE DASHBOARD PROJECT

TABLE OF CONTENTS

---

1. **Lets Recap**

   A brief overview of what Elasticsearch is can be found here?

2. **Develop a rough understanding of Web application serving. Namely wrt backend logs focusing on the components: Django/Gunicorn, and Celery**

   2.1. Nginx: An HTTP and Reverse Proxy Server

   2.2. Gunicorn: A WSGI HTTP server. Listens for connections

   2.3. Celery: A tool for asynchronous processing with Python

   Here's a cool brief article to check out on managing logs with Django, Gunicorn and Nginx. In this exercise you need not create the components from ground level but should you need to you can follow these steps

   Ingest pipelines

3. **Use existing log files (Django/Gunicorn, and Celery) to determine if there is a preferred way to ingest (with appropriate parsing) using Filebeat/Logstash**

   3.1. *If there are methods then document them with specific focus on what needs to be implemented*

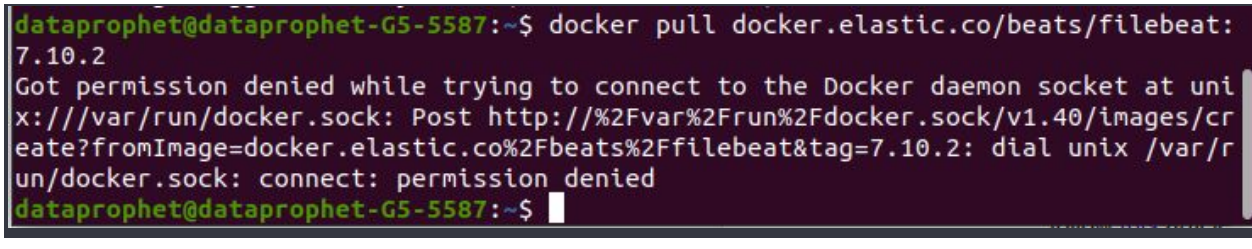
   When you use Filebeat modules with Logstash, you can use the ingest pipelines provided by Filebeat to parse the data. You need to load the pipelines into Elasticsearch and configure Logstash to use them. To load the ingest pipelines follow this article. No need to push that into logstash

   The different ways to ingest data

   - Kubernetes - You can use Filebeat Docker images on Kubernetes to retrieve and ship container logs.
     Q: What is Red Hat Openshift config?

   - Docker

     ```
     docker pull docker.elastic.co/beats/filebeat:7.10.2
     ```

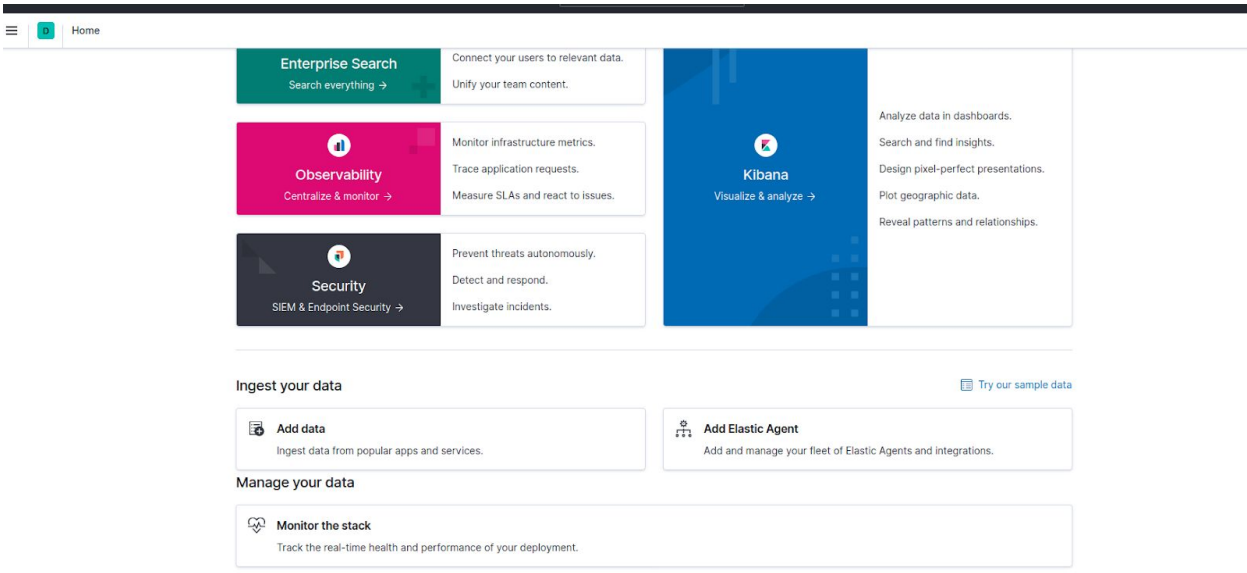
   - Cloud Foundry

- APT or YUM
- Download page

3.2. *If not we'll need to parse logs on the Kibana side*

Parsed logs with Kibana

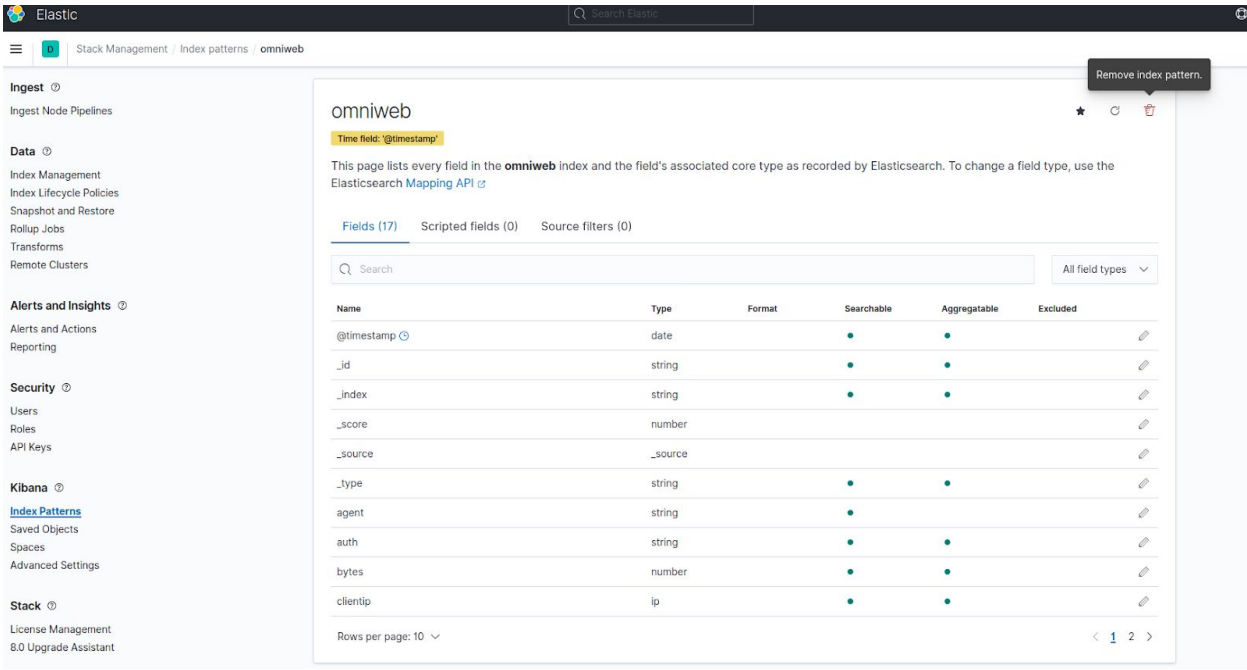
NB: User functionalities to load data aren't available for Username: dataprophet

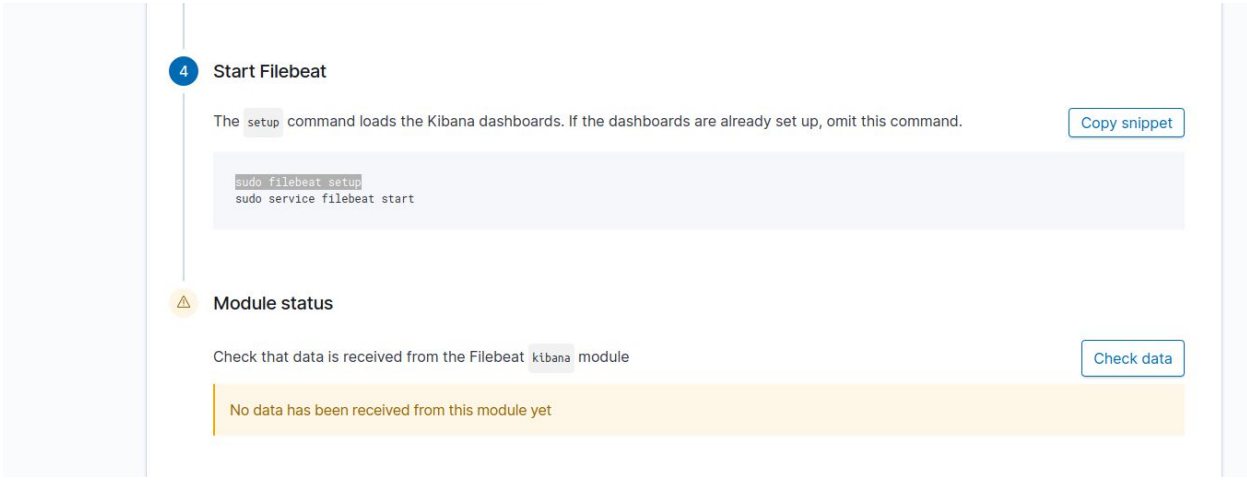


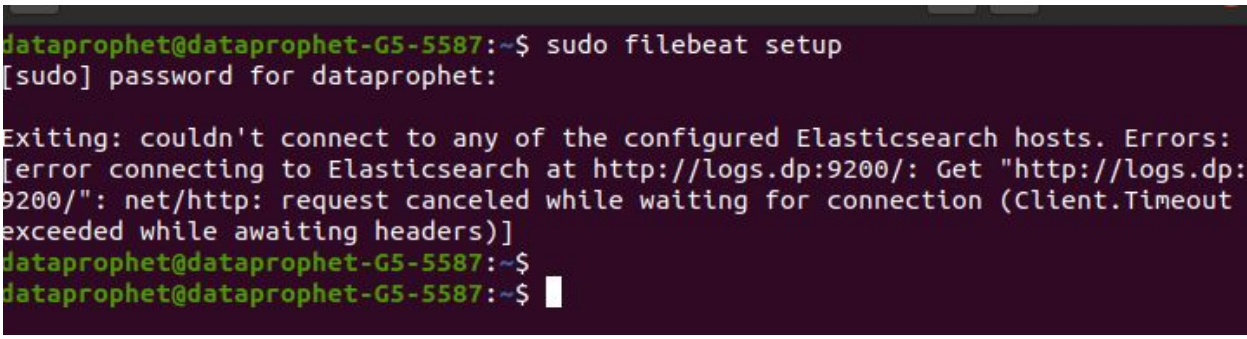
But they are there for username: tshepiso.moloko ensure to speak to the admin regarding creating this user profile and its functionalities. Make sure you are signed out.

Should you create an index pattern mistakenly you can delete it in the stack management

tab as shown below

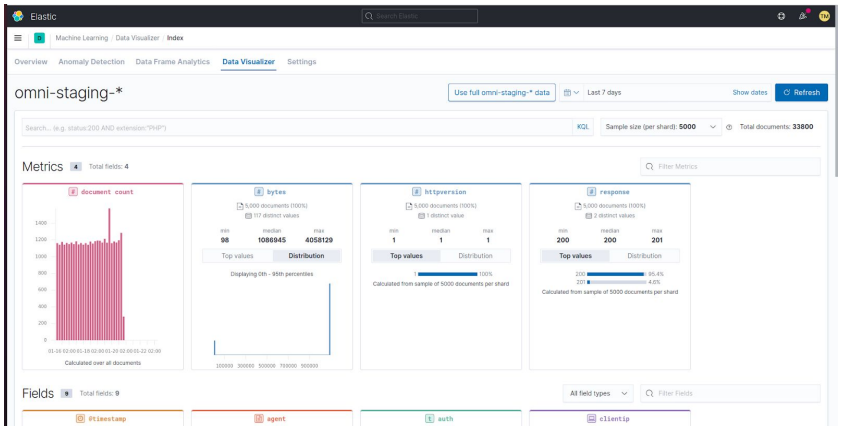






This has to do with configuring the actual web app

4. Explore the ingested log OMNI Web log data (NGinx, Django, Gunicorn, Celery)



4.1. *Save views that may be useful for further analysis. Aspects may include:*

4.2. *Usage activity*

4.3. *Errors*

Logs above 100MB

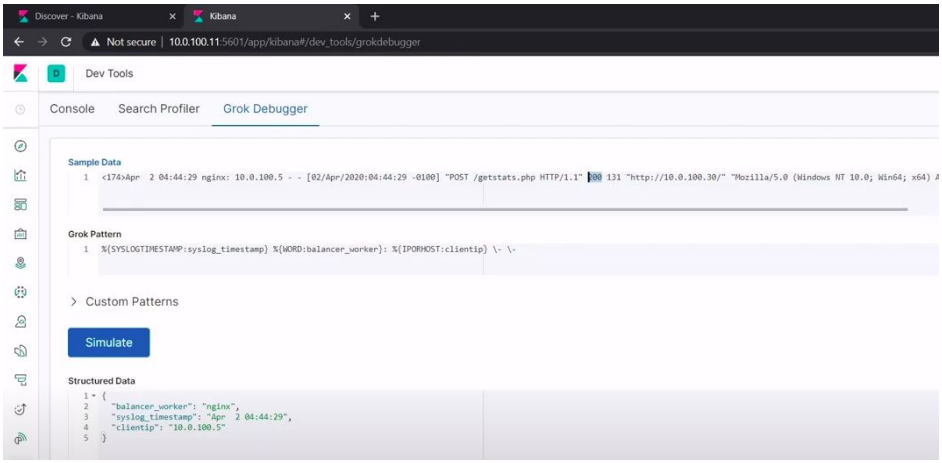
```
GET omni-staging-error
{
  "query": {
    "match_all": {}
  }
}
```

```
{
  "omni-staging-error" : {
    "aliases" : { },
    "mappings" : {
      "_meta" : {
        "created_by" : "ml-file-data-visualizer"
      },
      "properties" : {
        "@timestamp" : {
          "type" : "date"
        },
        "message" : {
          "type" : "text"
        }
      }
    },
    "settings" : {
      "index" : {
        "routing" : {
          "allocation" : {
            "include" : {
              "_tier_preference" : "data_content"
            }
          }
        },
        "number_of_shards" : "1",
        "provided_name" : "omni-staging-error",
        "creation_date" : "1611562563810",
        "number_of_replicas" : "1",
        "uuid" : "L4pZ_TTPTVOHPGM5CfeJBw",
        "version" : {
          "created" : "7100099"
        }
      }
    }
  }
}
```

4.4. *Request tracking and latency*

5. How to Extract Patterns with the Logstash Grok Filter

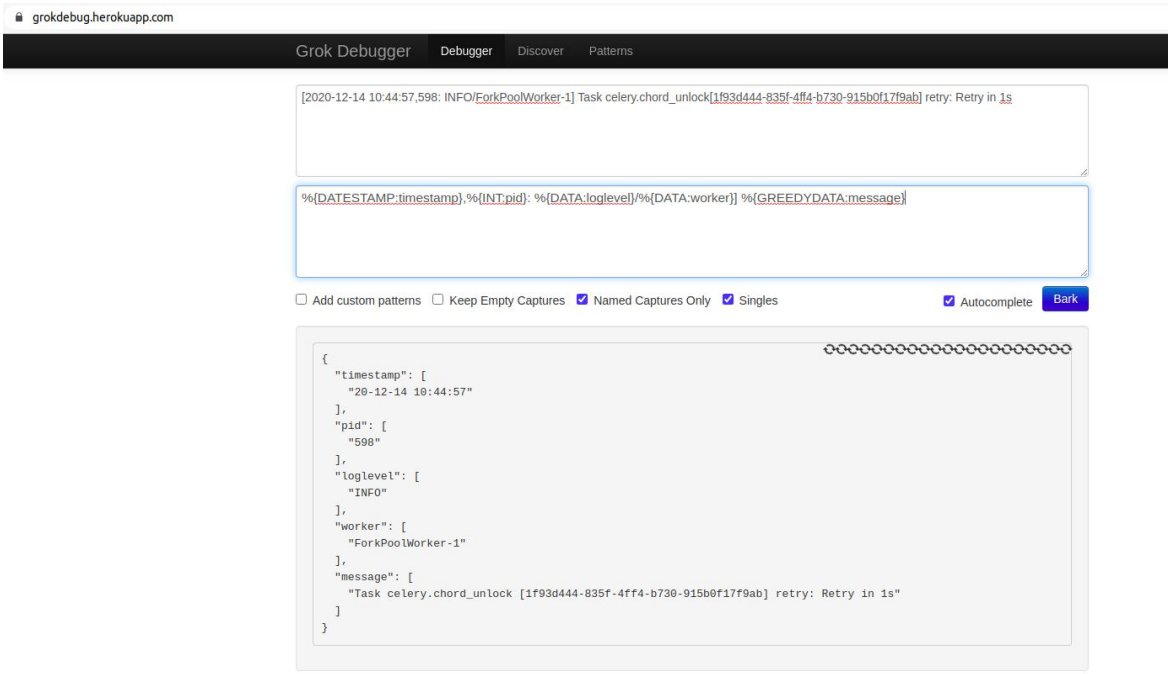
- Introduction doc to go through here
- Structured data with valuable fields
- Data transformation and normalization to classify
- System and Web Server logs
- Grok patterns that are labels and regex(regular expression)
  Logstash config example
  Use this grok debugger to get the correct pattern



- Oniguruma?

6. Grok Debugging

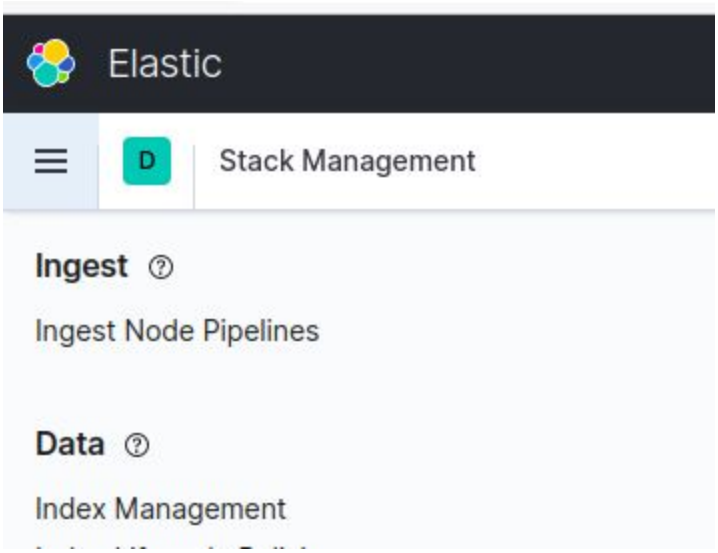
- Identify the grok pattern as per field you want to get



- Check for different test cases of the data set that has that ingested

- *Issue when not mapping:*

- Ensure that the same edits you made in Index Node Pipelines is also in your Index Manager under the Data tab.
- A way of doing so is by cloning the already existing index patten



Cloning API index

- Index Management template
- When ingesting with Logstash no need to do the mapping

Elastic search pipeline config

```
}
```

We can now continue building out the pattern step by step. Once we have successfully parsed out all fields, we can remove the message field so we do not hold the same data twice. We can do this using the remove_field directive, which only runs if the parsing was successful, resulting in the following filter block.

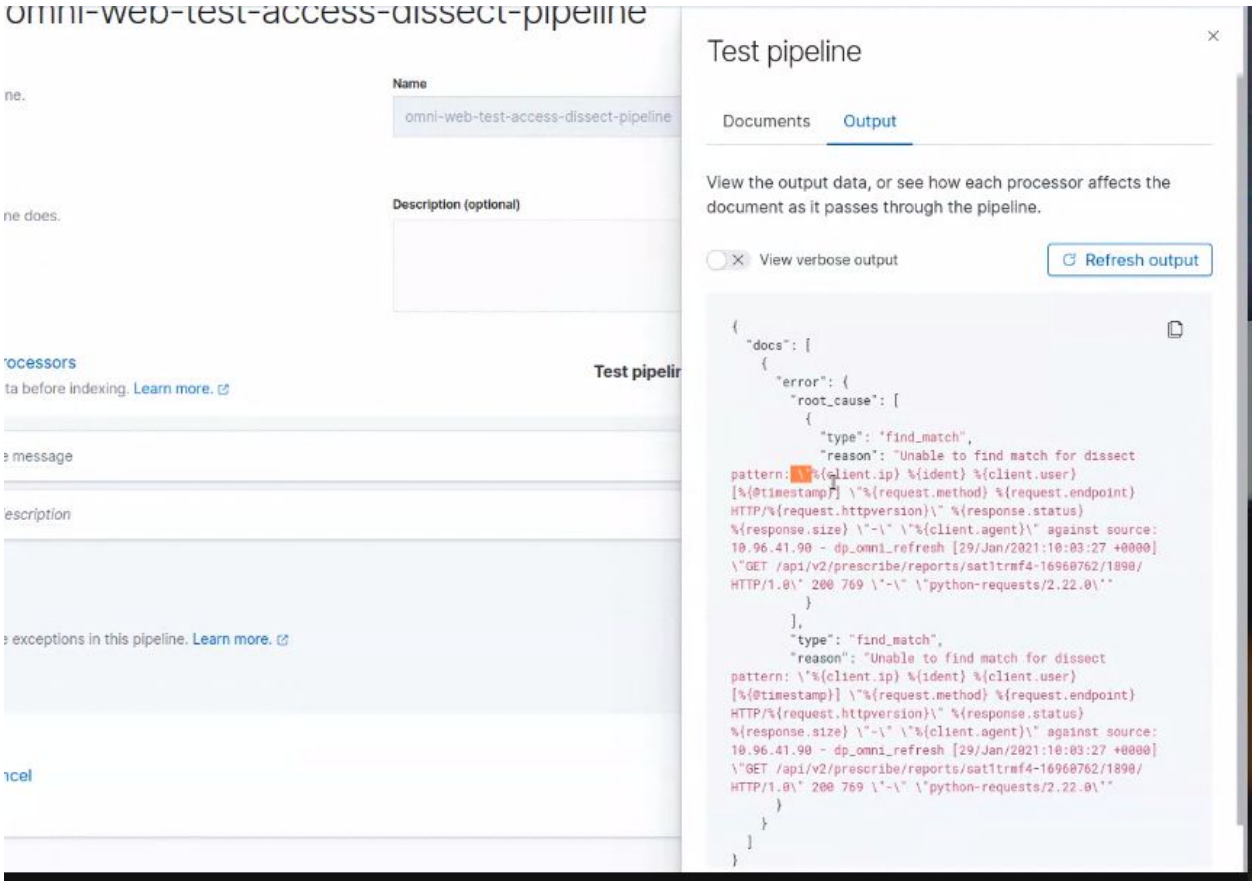
```
filter {
 dissect {
   mapping => {
     "message" => "%{timestamp->} %{duration} %{client_address} %{cache_result}/%{status_code}
%{bytes} %{request_method} %{url} %{user} %{hierarchy_code}/%{server} %{content_type}"
   }
   remove_field => ["message"]
  }
}
```

When run against the sample data, the first record looks as follows:

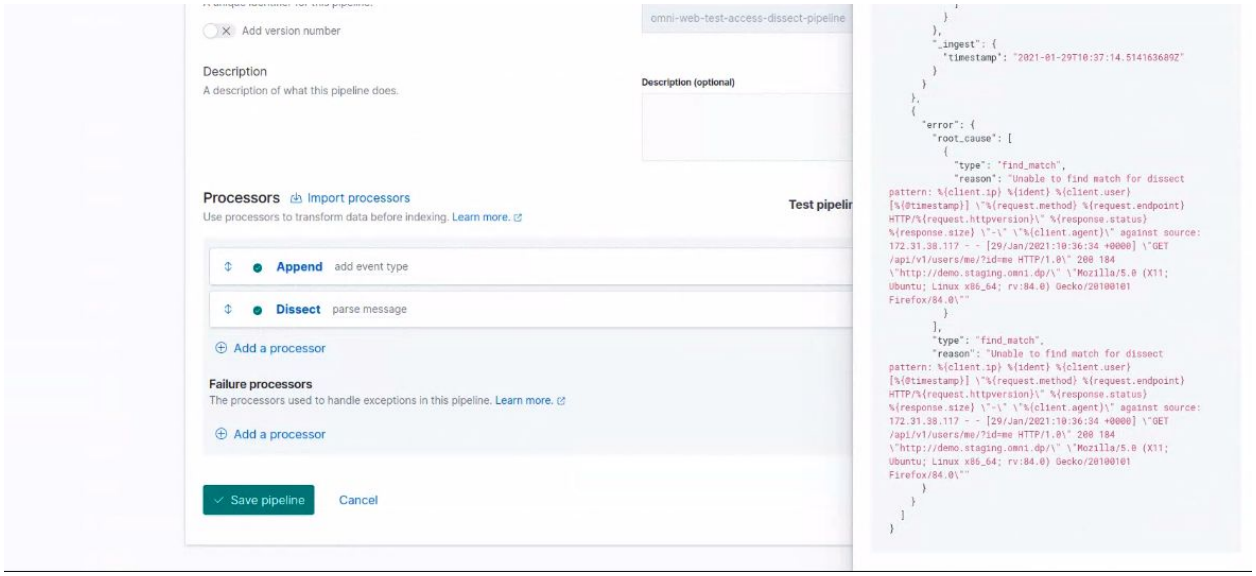
```
{
```

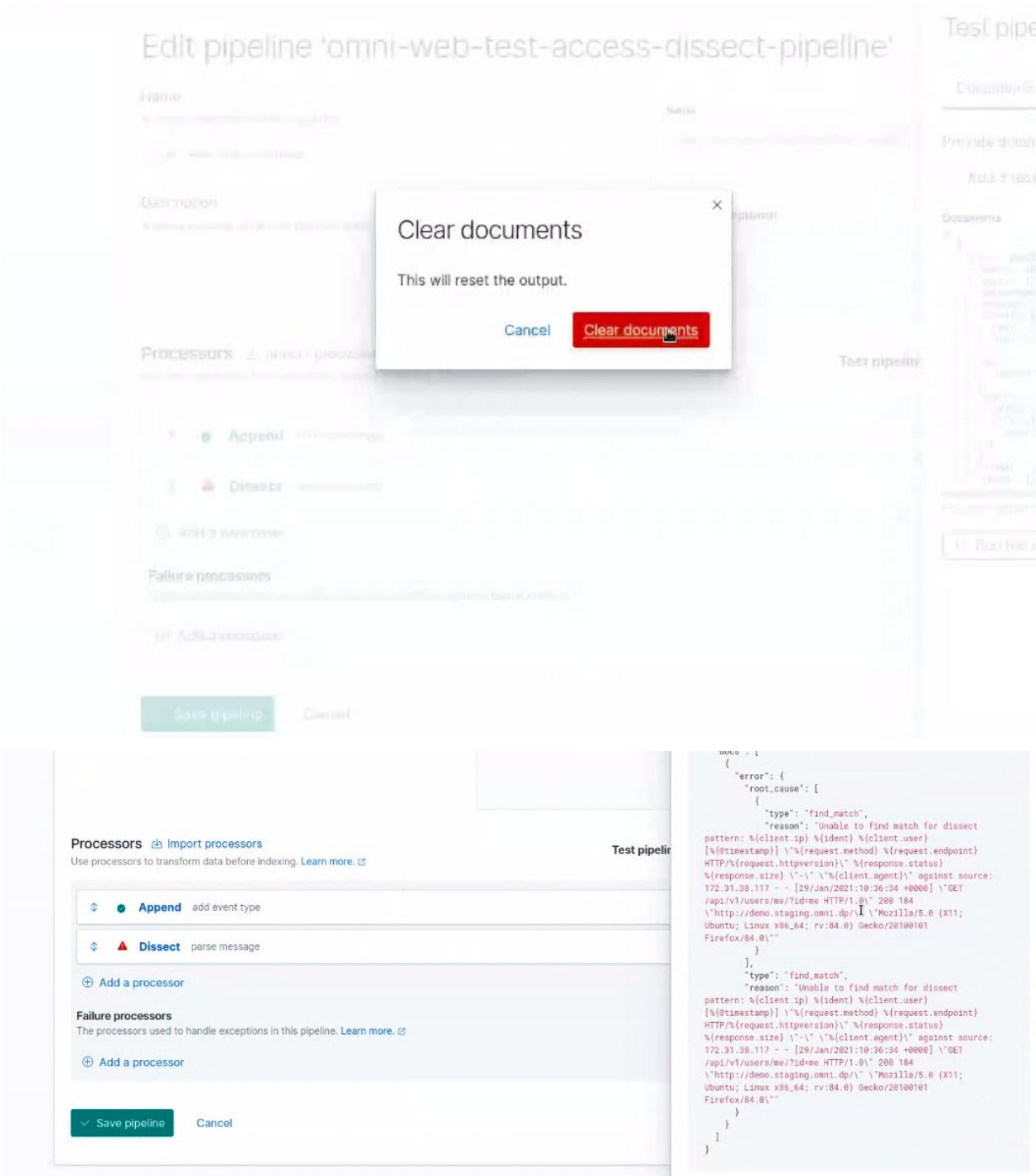
Logstash not able to process Elastisc node as quickly - for scalability on the future

If the pipeline doesnt process it uses the REMOVE to



How to know if we have tested all possible cases:





- To escape characters in your loglines make use of this doc

7. The difference between Grok patterns, CSV and Dissect patterns

Grok + CSV

```
filter {
  if [message] =~ "THREAT," {
    grok {
      match => { "message" => "\A<%{POSINT:priority}>%{SYSLOGTIMESTAMP:syslog_timestamp} %
{SYSLOGHOST:sysloghost} %{DATA:pan_fut_use_01},%{DATA:pan_rec_time},%{DATA:pan_serial_number},%
{DATA:pan_type},%{GREEDYDATA:message}\z" }
      overwrite => [ "message" ]
    }
    csv {
      separator => ","
      columns => ["pan_subtype", "pan_fut_use_02", "pan_gen_time", "pan_src_ip", "pan_dst_ip",
"pan_nat_src_ip", "pan_nat_dst_ip", "pan_rule_name
                                                Read More↓
```

Dissect

```
filter {
  if [message] =~ "THREAT," {
    dissect {
      mapping => {
        message => "<%{priority}>%{syslog_timestamp} %{+syslog_timestamp} %{+syslog_timestamp} %
{logsource} %{pan_fut_use_01},%{pan_rec_time},%{pan_serial_number},%{pan_type},%{pan_subtype},%
{pan_fut_use_02},%{pan_gen_time},%{pan_src_ip},%{pan_dst_ip},%{pan_nat_src_ip},%{pan_nat_dst_ip},%
{pan_rule_name},%{pan_src_user},%{pan_dst_user},%{pan_app},%{pan_vsys},%{pan_src_zone},%
{pan_dst_zone},%{pan_ingress_intf
                                                Read More↓
```

Output

```
output {
  counter { warmup => 60 }
}
```

The output is a new output that I built - it captures the number of events and the start time of receiving the first event and

8. Create visualizations and dashboards based on the explored views

8.1. *Visualizations and dashboards need to serve a specific purpose. The best approach may be to keep in mind multiple different dashboards with different outcomes, for example:*

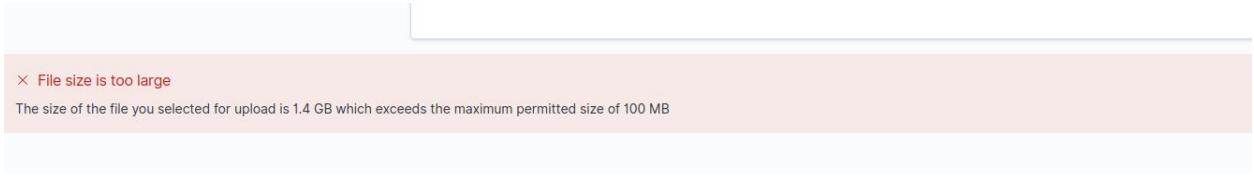
8.2. *User activity --- tracking daily activity of users to and in the PRESCRIBE application*

8.3. *Report generation errors --- identifying error events from Celery workers and providing easy drilldown into event details*

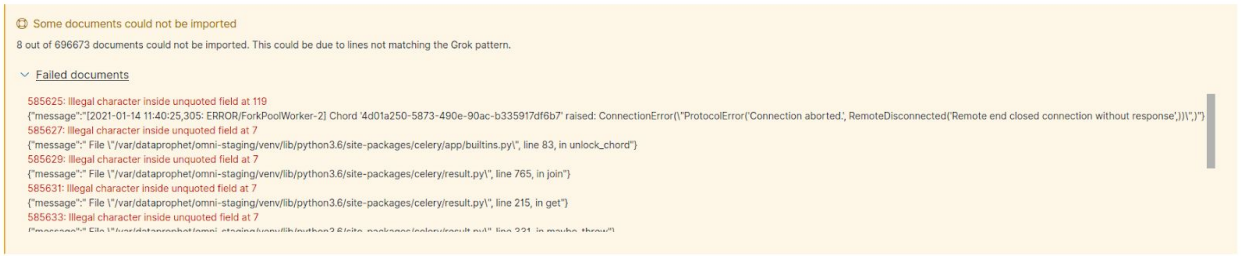
8.4. *Web Application errors --- identifying error events from Django/Gunicorn and Nginx and providing easy drilldown into event details*

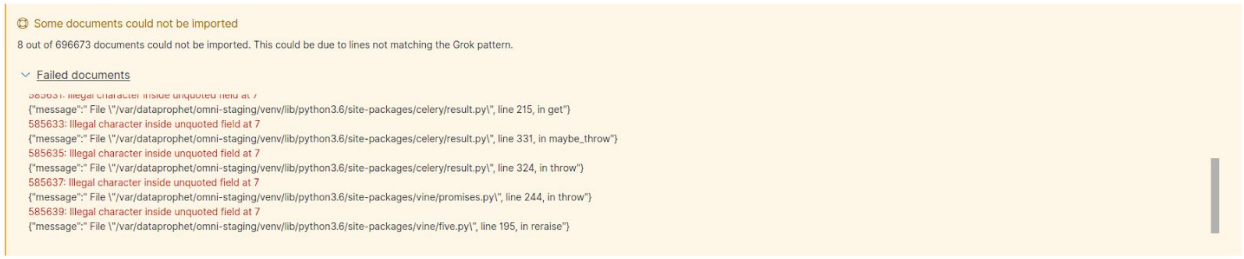
ISSUES WITH LOADING LOGS

- Limitation of size of logs that can be loaded

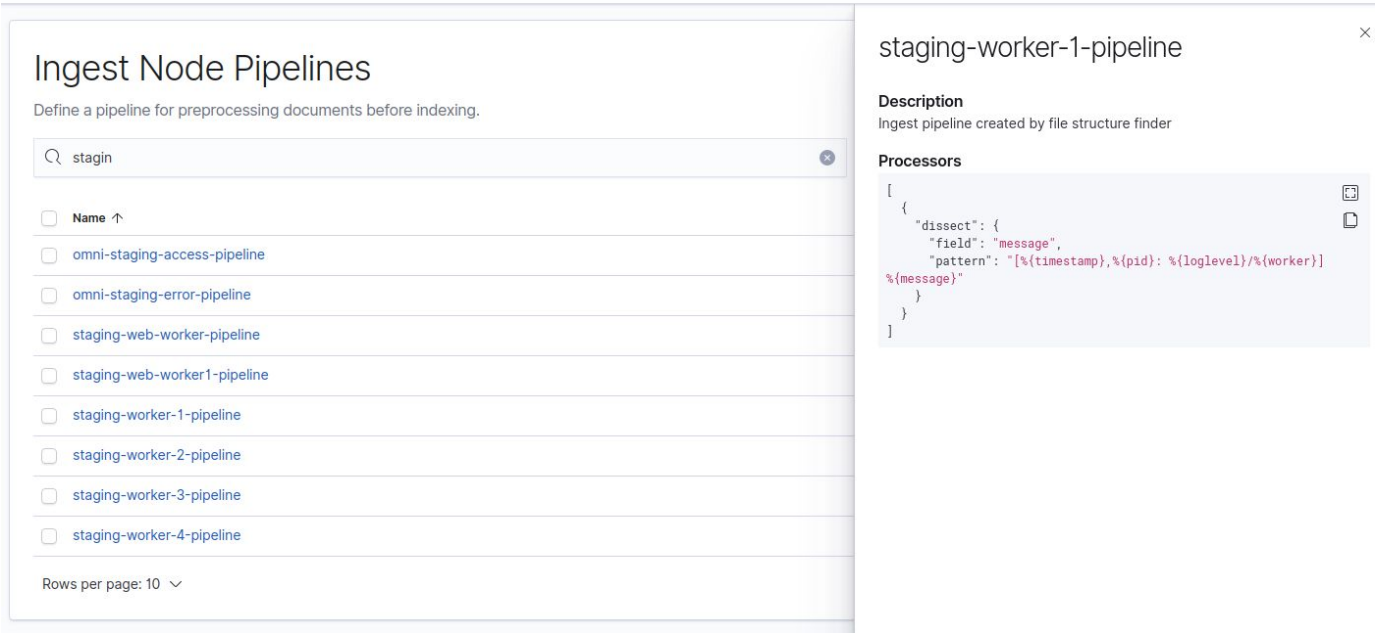


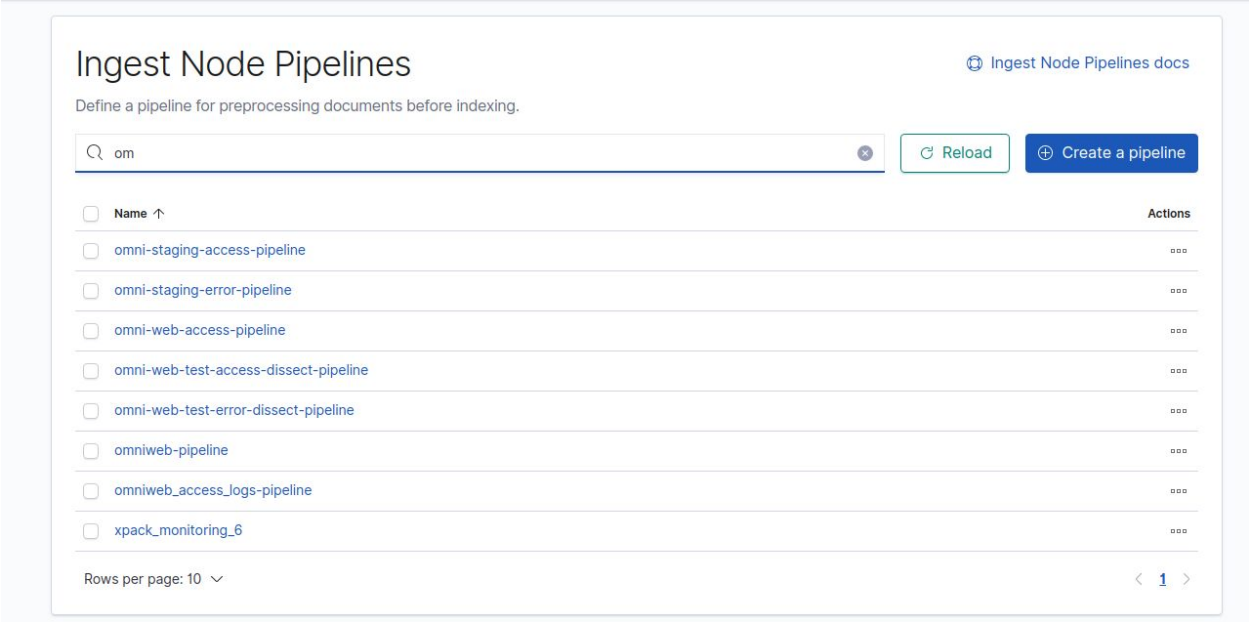
- Uploading multiple logs
  An index pattern can match a single source, for example, filebeat-4-3-22, or **multiple** data sources, filebeat-*.
- Fortunately kibana does show some errors and reasons for some documents that couldn't be uploaded. When aggregating be mindful of these.





Clean up

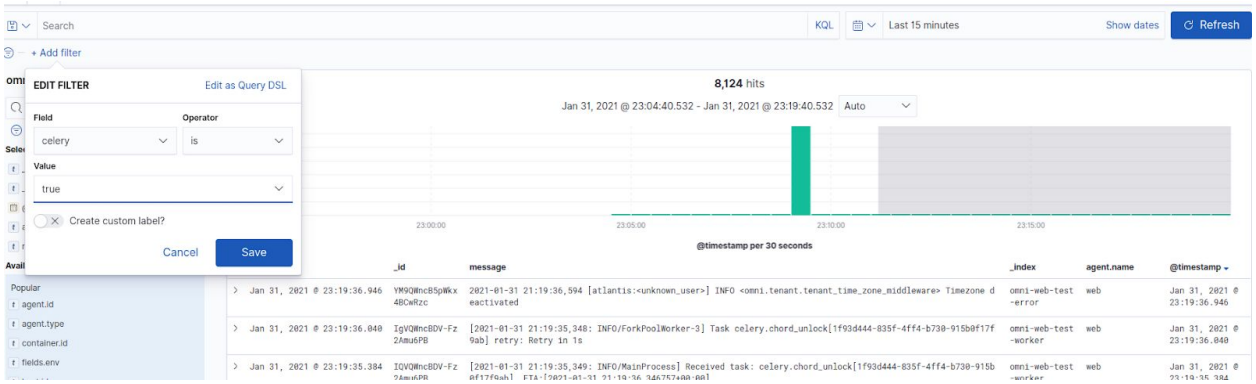




Grok pattern:

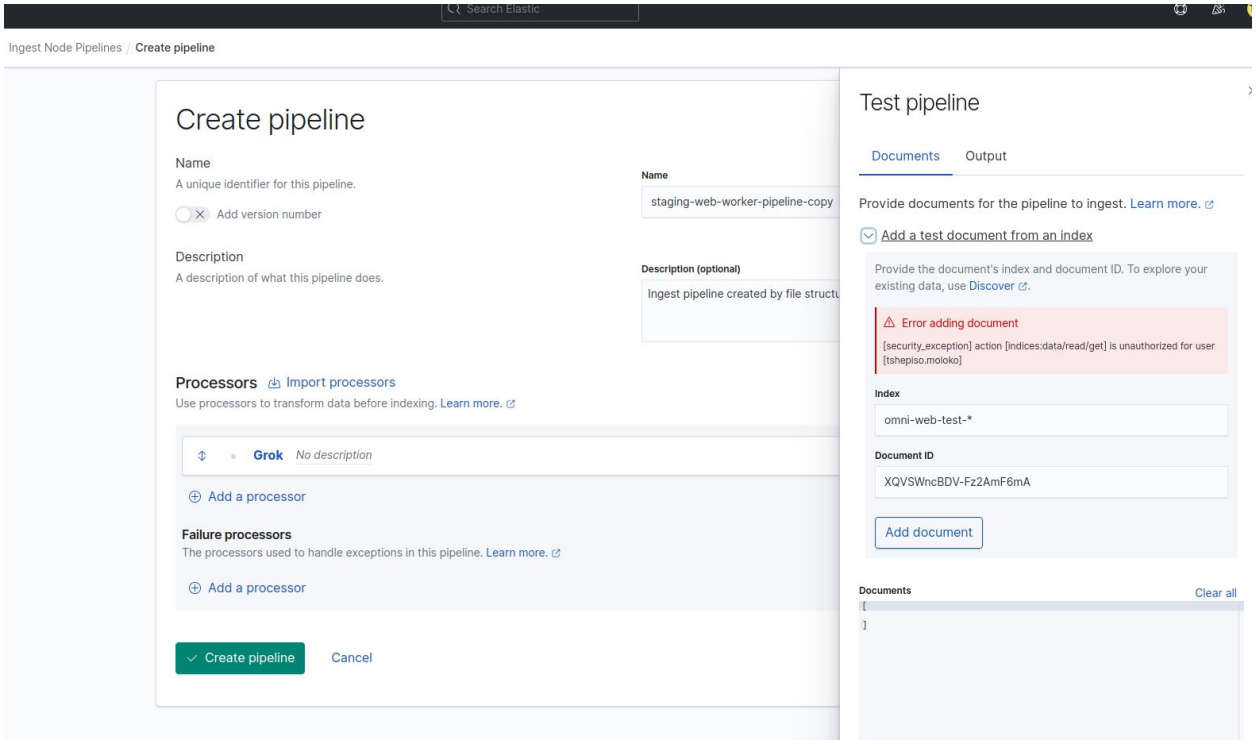
```
%{DATESTAMP:timestamp},%{INT:pid}: %{DATA:loglevel}/%{DATA:worker}]
%{GREEDYDATA:message}
```

Testing Staging Workers with live logs



Test all test case

Question



Testing grok pattern with the new logs

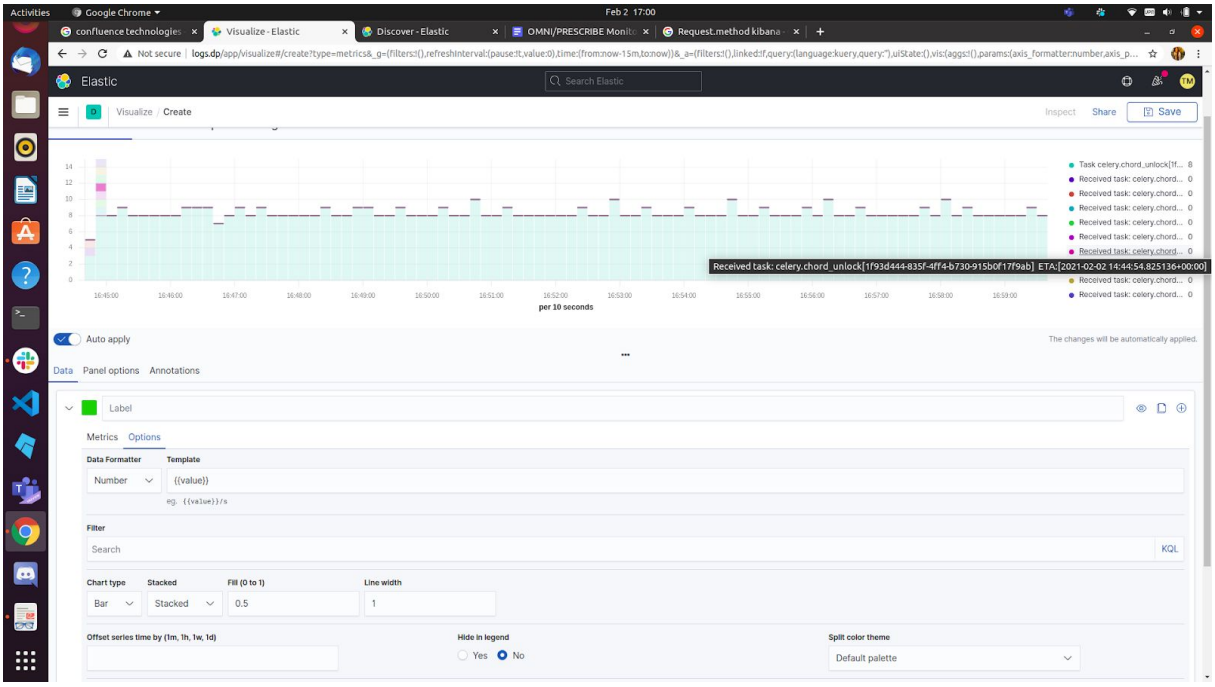
DASHBOARDING - what to look for in monitoring dashboard distributed web applications

Naming Format: Name of breakdown [Source]

Staging Logs:

- Container.id:The *source of information*.
- Process.pid:correlate metrics information
- Log.file.path:
- Log.offset: The file offset the reported line starts at
- Message: Received task/Retry
- Omni.worker_id / Log.file.path:
- @timestamp:

1. Breakdown of each worker (percentages)
2. When do we get specific messages for each worker[Retry / Received]
3. What are the Responses for each worker?
4. Show specific occurrences...
5. Which worker and when had the highest /lowest user activity in a specific period?
6. How much time per process based on the main
7. Tracking tasks from each worker



Access Logs

- `_id:journey`
- `Request.method:` Defaults to OPTIONS, HEAD, GET, POST, PUT, DELETE
- `Log.offset:`
- `@timestamp:`

1. When do users log in events access the application?
   -per tenant
   -across all tenants
2. What request methods are on the application- coupled with endpoints
3. What responses from the hits above of the application
4. When do those requests occur?
5. What are the peak or low points/ranges of these requests?
6. Most in demand features

Error Logs:

- **Log.file.path:**
- **Request.user:**
- **Request.tenant:**
- **Process.pid:**
- **Log.offset:**
- **Message:**
- **@timestamp:**

1. What errors do users get? Show the different times and occurrences
2. Error rate at a specific time and then show the log line that caused that specific error
3. When do they get those errors?
4. How often do we get the specific errors in the last{hour/day/month} period?

Check the SRE books

ENSURING THAT DASHBOARD HAS A THEME AND DIRECTION

- MONITORING:

  -->Alerts
  -->Tickets
  -->Logging
  -->Emergency response
  -->Change Management

-->Demand Forecasting & Planning
-->Provisioning
-->Efficiency and Performance

Keep the Golden 4 in mind when building your dashboard:

These mainly refer to metrics and are common themes to use. There

Latency

The time it takes to service a request. It's important to distinguish between the latency of successful requests and the latency of failed requests.
For example, an HTTP 500 error triggered due to loss of connection to a database or other critical backend might be served very quickly;

Traffic

A measure of how much demand is being placed on your system, measured in a high-level system-specific metric.
For a web service, this measurement is usually HTTP requests per second, perhaps broken out by the nature of the requests (e.g., static versus dynamic content).
For a key-value storage system, this measurement might be transactions and retrievals per second.

Errors

The rate of requests that fail, either explicitly (e.g., HTTP 500s), implicitly (for example, an HTTP 200 success response, but coupled with the wrong content), or by policy (for example, "If you committed to one-second response times, any request over one second is an error").
Where response codes are insufficient to express all failure conditions, secondary (internal) protocols may be necessary to track partial failure modes.
Monitoring these cases can be drastically different: catching HTTP 500s at your load balancer can do a decent job of catching all completely failed requests, while only end-to-end system tests can detect that you're serving the wrong content.

Saturation

How "full" your service is. A measure of your system fraction, emphasizing the resources that are most constrained (e.g., in a memory-constrained system, show memory; in an I/O-constrained system, show I/O). Note that many systems degrade in performance before they achieve 100% utilization, so having a utilization target is essential.

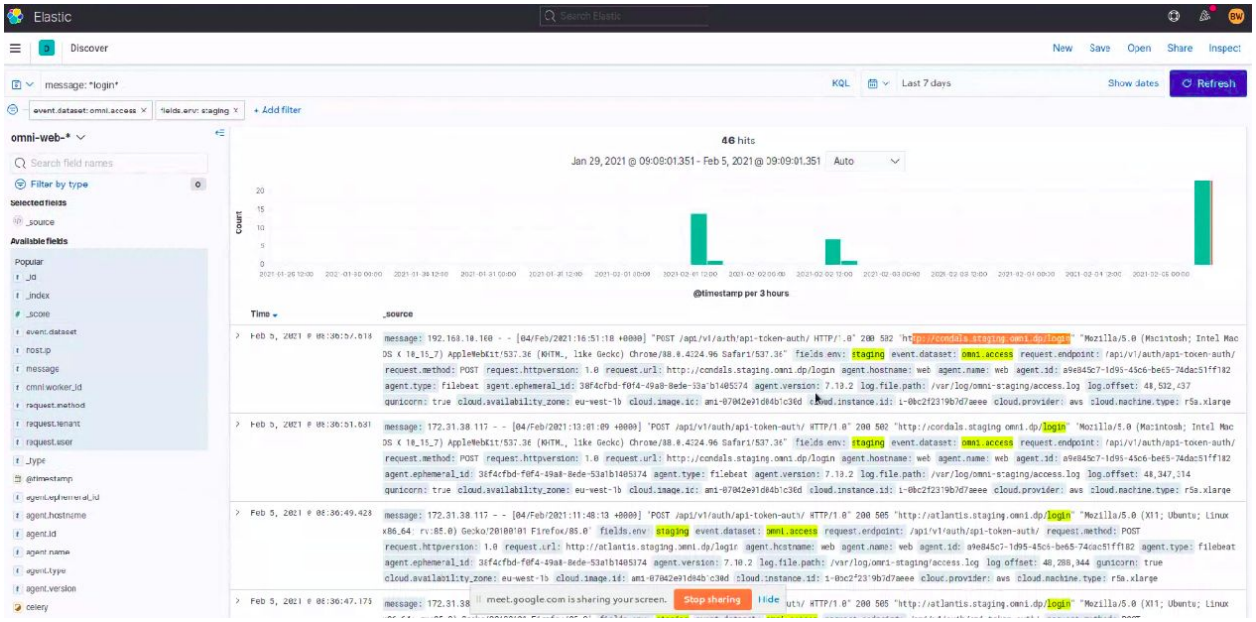
```
2021-02-08T07:25:58.757Z    DEBUG    [elasticsearch] elasticsearch/client.go:414 Bulk item insert failed (i=0, status=500): {"type":"find_match","reason":"Unable to find match for dissect pattern: [%{event.created}] [%{process.pid}] [%{log.level}] %{message} against source: /var/dataprophet/omni-staging/venv/lib/python3.6/site-packages/django/db/models/fields/__init__.py:1427: RuntimeWarning: DateTimeField RawValue.timestamp received a naive datetime (2021-01-29 12:11:34) while time zone support is active.\n  RuntimeWarning)\n/var/dataprophet/omni-staging/venv/lib/python3.6/site-packages/django/db/models/fields/__init__.py:1427: RuntimeWarning: DateTimeField RawValue.timestamp received a naive datetime (2021-02-05 12:11:34) while time zone support is active.\n  RuntimeWarning)"}
```

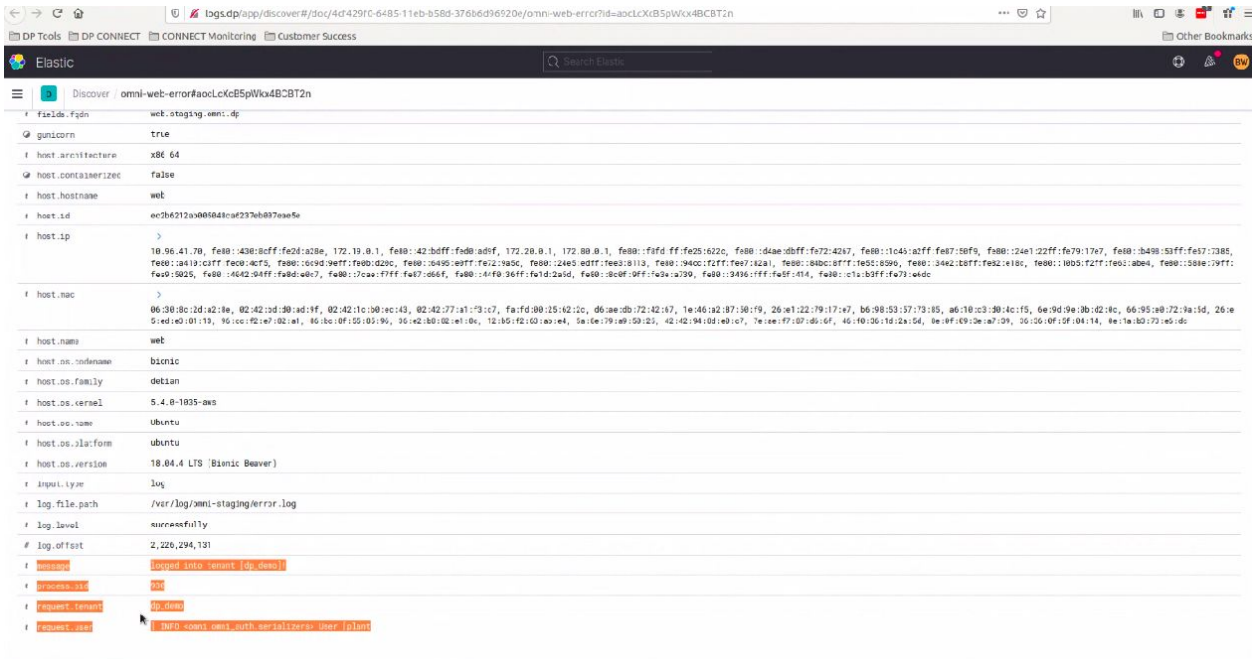
- When sources are broken - sudo conditions to fix this error
- Not treating that as multilines

ISSUE WITH ERROR LOGS

When aggregating the access logs on the Prescribe application we filter the logs streamed by the following keyword login event within the message:

`172.31.38.117 - - [08/Feb/2021:06:53:21 +0000] "GET /admin/login/?next=/admin/ HTTP/1.0" 200 1939 "-" "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/88.0.4324.96 Safari/537.36"`





Parsing is not occurring correctly as seen in the above log line

```
2021-02-05 07:26:30,863 [dp_demo:1] INFO <omni.omni_auth.serializers> User [plant] successfully logged into tenant [dp_demo]!
```

It should be in this format for the application logs that we see coming from ONMI:

https://github.com/DataProphet/omni_web/blob/66987463d059af207198f09536e0f09299a781f2/omni-backend/config/settings.py#L306

```
"%(asctime)s [%(tenant_name)s:%(user_name)s] %(levelname).4s <%(name)s> %(message)s"
```

Note: Simulation in the pipelines doesn't make the change the log line it tests it.