

OMNI/PRESCRIBE MONITORING USAGE DASHBOARD PROJECT

TABLE OF CONTENTS

1. LETS RECAP
 2. Develop a rough understanding of Web application serving. Namely wrt backend logs focusing on the components: Django/Gunicorn, and Celery 1
 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 1
 - Grok Debugger
 4. Explore the ingested log OMNI Web log data (NGinx, Django, Gunicorn, Celery) 1
 5. Create visualizations and dashboards based on the explored views
 6. Dashboarding - What to look for in monitoring dashboard distributed web applications
-

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
```

```
dataprophet@dataprophet-G5-5587:~$ docker pull docker.elastic.co/beats/filebeat:7.10.2
Got permission denied while trying to connect to the Docker daemon socket at unix:///var/run/docker.sock: Post http://%2Fvar%2Frun%2Fdocker.sock/v1.40/images/create?fromImage=docker.elastic.co%2Fbeats%2Ffilebeat&tag=7.10.2: dial unix /var/run/docker.sock: connect: permission denied
dataprophet@dataprophet-G5-5587:~$
```

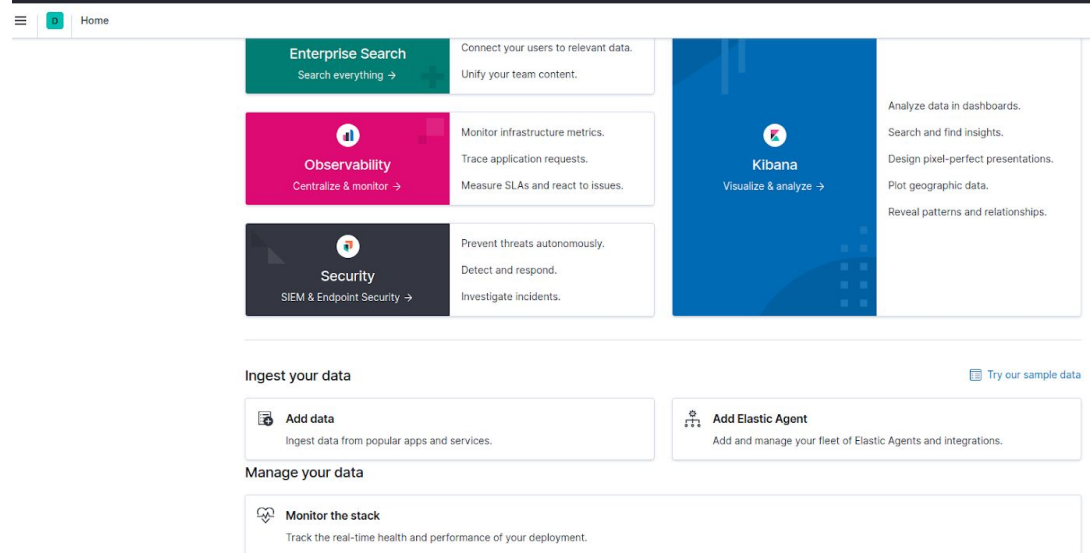
- Cloud Foundry

- 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

Elastic

Stack Management / Index patterns / omniweb

Ingest

Ingest Node Pipelines

Data

Index Management
Index Lifecycle Policies
Snapshot and Restore
Rollup Jobs
Transforms
Remote Clusters

Alerts and Insights

Alerts and Actions
Reporting

Security

Users
Roles
API Keys

Kibana

[Index Patterns](#)
Saved Objects
Spaces
Advanced Settings

Stack

License Management
8.0 Upgrade Assistant

omniweb

Time field: '@timestamp'

This page lists every field in the **omniweb** index and the field's associated core type as recorded by Elasticsearch. To change a field type, use the [Elasticsearch Mapping API](#).

Fields (17) Scripted fields (0) Source filters (0)

Search

All field types

Name	Type	Format	Searchable	Aggregatable	Excluded
@timestamp	date				
_id	string				
_index	string				
_score	number				
_source	_source				
_type	string				
agent	string				
auth	string				
bytes	number				
clientip	ip				

Rows per page: 10

4 Start Filebeat

The `setup` command loads the Kibana dashboards. If the dashboards are already set up, omit this command.

[Copy snippet](#)

```
sudo filebeat setup
sudo service filebeat start
```

Module status

Check that data is received from the Filebeat `ktbana` module

[Check data](#)

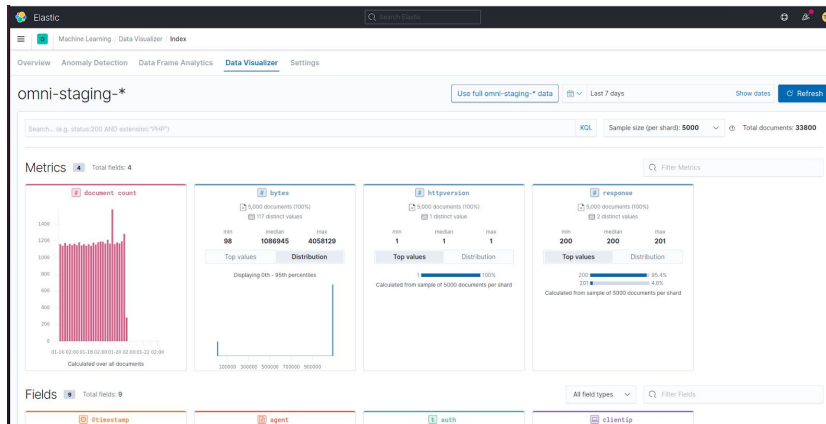
No data has been received from this module yet

```
dataprophet@dataprophet-G5-5587:~$ sudo filebeat setup
[sudo] password for dataprophet:

Exiting: couldn't connect to any of the configured Elasticsearch hosts. Errors:
[error connecting to Elasticsearch at http://logs.dp:9200/: Get "http://logs.dp:
9200/": net/http: request canceled while waiting for connection (Client.Timeout
exceeded while awaiting headers)]
dataprophet@dataprophet-G5-5587:~$
dataprophet@dataprophet-G5-5587:~$
```

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

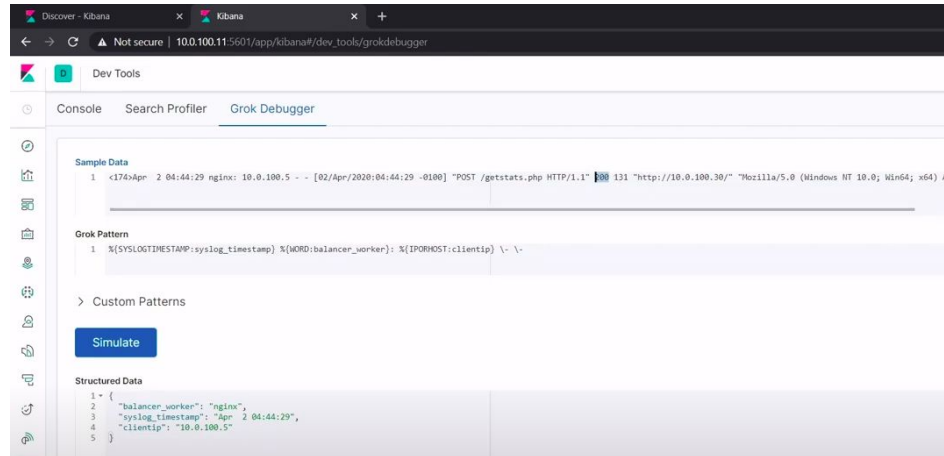
```
GET omni-staging-error/
{
  "query": {
    "match_all": {}
  }
}
```

```
{
  "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",
        "uid": "L4pZ-ITPTVQHPCMS4feJBw",
        "version": {
          "created": "7100099"
        }
      }
    }
  }
}
```

4.4. Request tracking and latency

5. How to Extract Patterns with the Logstash Grok Filter

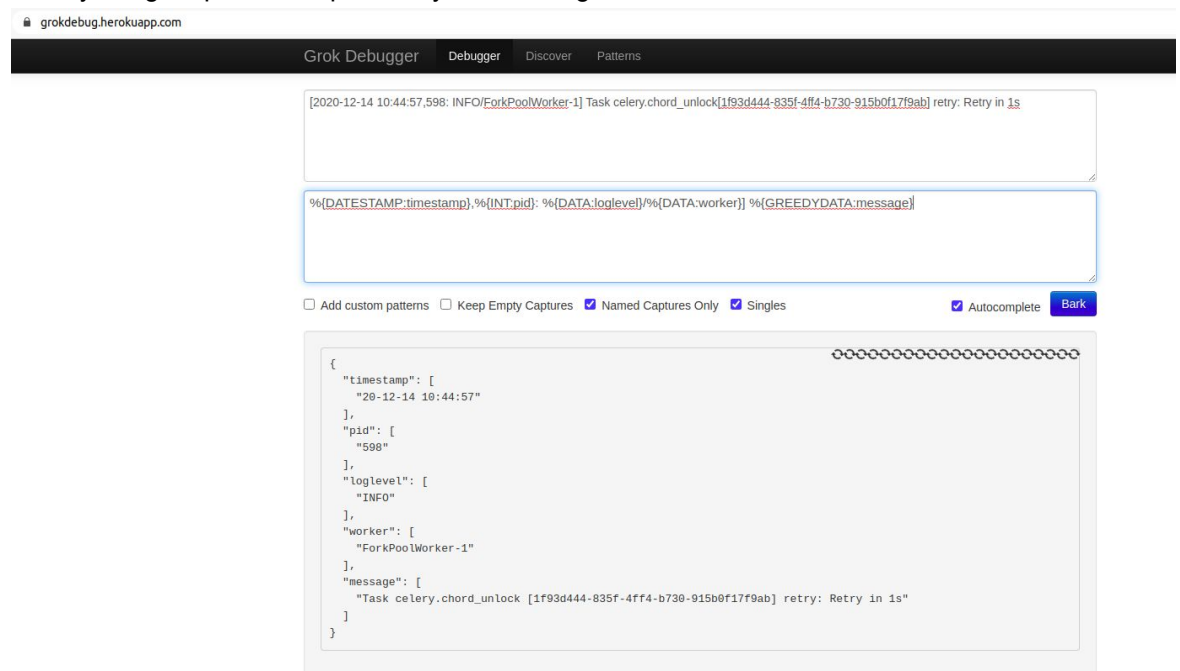
- Introduction doc to go through [here](#)
 - Structured data with valuable fields
 - Data transformation and normalization to classify
 - System and Web Server logs
 - Grok patterns that are labels and regex(regular expression)
- Logstash config [example](#)
- Use [this](#) grok debugger to get the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



- Oniguruma?

6. Grok Debugging

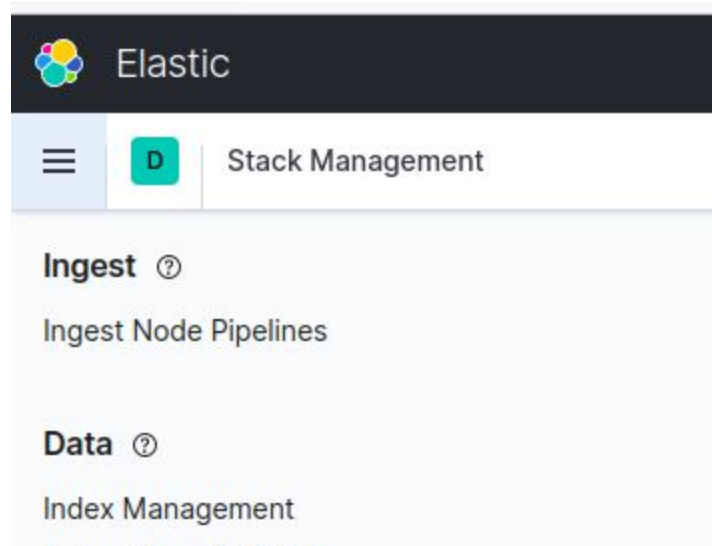
- Identify the grok pattern as per field you want to get



- 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 pattern



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

omni-web-test-access-dissect-pipeline

ne.

ne does.

Processors

ta before indexing. [Learn more.](#)

message

Description

exceptions in this pipeline. [Learn more.](#)

1cel

Name

omni-web-test-access-dissect-pipeline

Description (optional)

Test pipeline

Test pipeline

Documents **Output**

View the output data, or see how each processor affects the document as it passes through the pipeline.

☐ View verbose output

[Refresh output](#)

```
{
  "docs": [
    {
      "error": {
        "root_cause": [
          {
            "type": "find_match",
            "reason": "Unable to find match for dissect
pattern: '%(client.ip) %(ident) %(client.user)
[%(@timestamp)]' '%(request.method) %(request.endpoint)
HTTP/%(request.httpversion)'" %(response.status)
%(response.size) '-' '-' '%(client.agent)'" against source:
10.96.41.90 - dp_omni_refresh [29/Jan/2021:10:03:27 +0000]
'GET /api/v2/prescribe/reports/satltmf4-16960762/1890/
HTTP/1.0' 200 769 '-' '-' 'python-requests/2.22.0'"
          }
        ],
        "type": "find_match",
        "reason": "Unable to find match for dissect
pattern: '%(client.ip) %(ident) %(client.user)
[%(@timestamp)]' '%(request.method) %(request.endpoint)
HTTP/%(request.httpversion)'" %(response.status)
%(response.size) '-' '-' '%(client.agent)'" against source:
10.96.41.90 - dp_omni_refresh [29/Jan/2021:10:03:27 +0000]
'GET /api/v2/prescribe/reports/satltmf4-16960762/1890/
HTTP/1.0' 200 769 '-' '-' 'python-requests/2.22.0'"
      }
    }
  ]
}
```

How to know if we have tested all possible cases:

☐ Add version number

Description

A description of what this pipeline does.

Processors [Import processors](#)

Use processors to transform data before indexing. [Learn more.](#)

Append add event type

Dissect parse message

Add a processor

Failure processors

The processors used to handle exceptions in this pipeline. [Learn more.](#)

Add a processor

Save pipeline

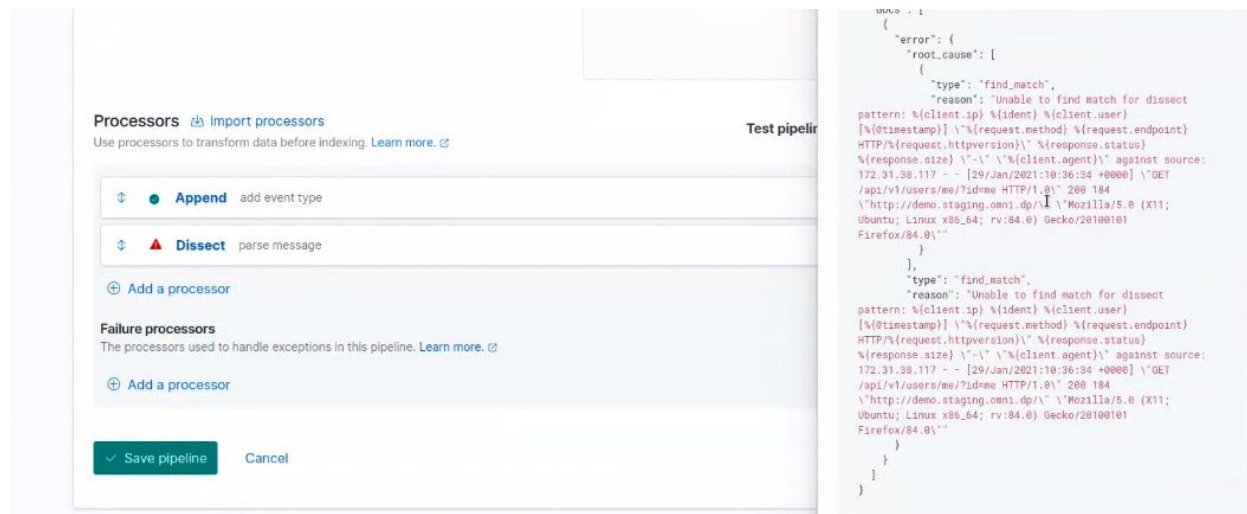
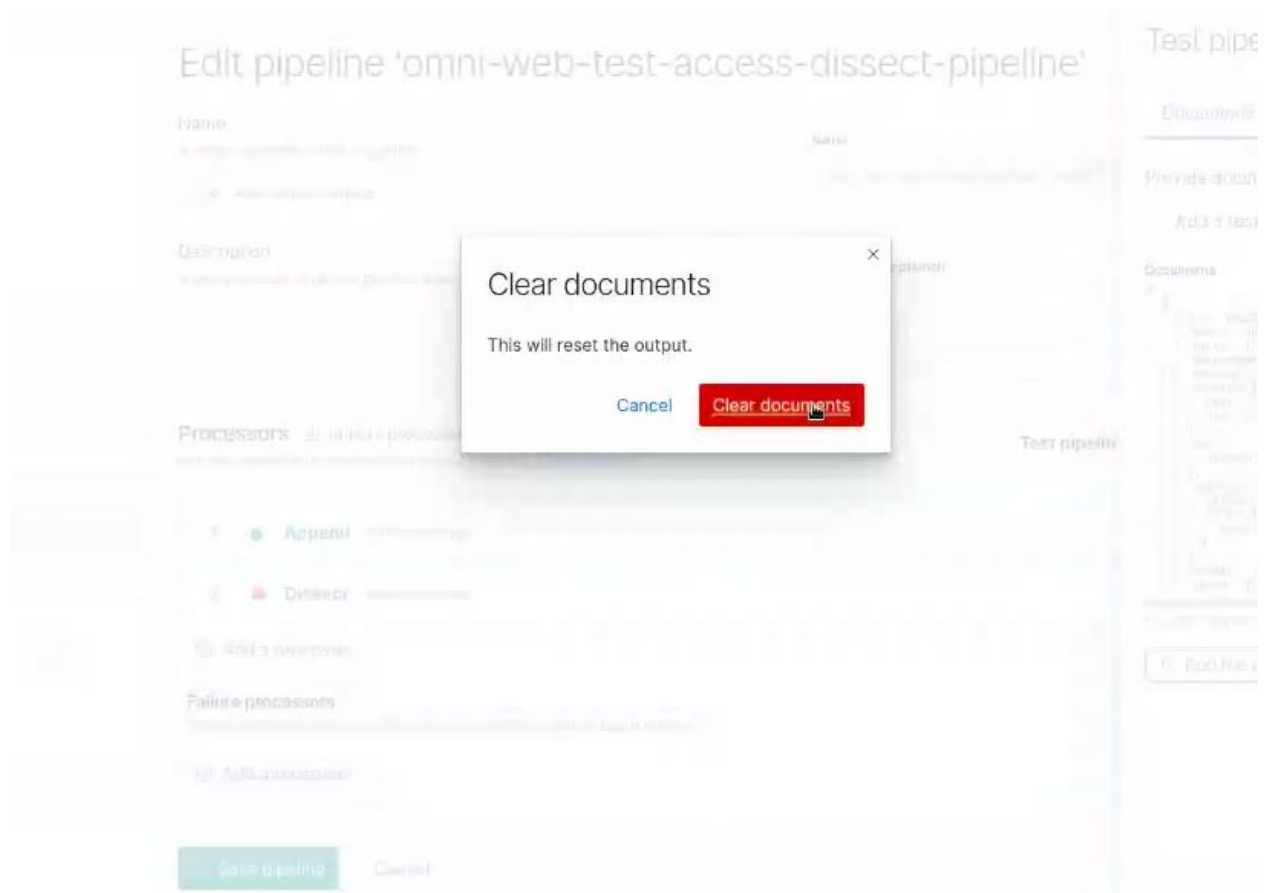
Cancel

omni-web-test-access-dissect-pipeline

Description (optional)

Test pipeline

```
{
  "ingest": {
    "timestamp": "2021-01-29T10:07:14.514163689Z"
  },
  "error": {
    "root_cause": [
      {
        "type": "find_match",
        "reason": "Unable to find match for dissect
pattern: '%(client.ip) %(ident) %(client.user)
[%(@timestamp)]' '%(request.method) %(request.endpoint)
HTTP/%(request.httpversion)'" %(response.status)
%(response.size) '-' '-' '%(client.agent)'" against source:
172.31.38.117 - - [29/Jan/2021:10:36:34 +0000] 'GET
/api/v1/users/me/15d-me HTTP/1.0' 200 184
'http://demo.staging.omni.dp/' 'Mozilla/5.0 (X11;
Ubuntu; Linux x86_64; rv:84.0) Gecko/20100101
Firefox/84.0'"
      }
    ],
    "type": "find_match",
    "reason": "Unable to find match for dissect
pattern: '%(client.ip) %(ident) %(client.user)
[%(@timestamp)]' '%(request.method) %(request.endpoint)
HTTP/%(request.httpversion)'" %(response.status)
%(response.size) '-' '-' '%(client.agent)'" against source:
172.31.38.117 - - [29/Jan/2021:10:36:34 +0000] 'GET
/api/v1/users/me/15d-me HTTP/1.0' 200 184
'http://demo.staging.omni.dp/' 'Mozilla/5.0 (X11;
Ubuntu; Linux x86_64; rv:84.0) Gecko/20100101
Firefox/84.0'"
      }
    ]
  }
}
```

- 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

× File size is too large

The size of the file you selected for upload is 1.4 GB which exceeds the maximum permitted size of 100 MB

- 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.

⚠ Some documents could not be imported

8 out of 696673 documents could not be imported. This could be due to lines not matching the Grok pattern.

▼ Failed documents

585625: illegal character inside unquoted field at 119

("message":"[2021-01-14 11:40:25,305: ERROR/ForkPoolWorker-2] Chord '4d01a250-5873-490e-90ac-b335917df6b7' raised: ConnectionError('ProtocolError('Connection aborted', RemoteDisconnected('Remote end closed connection without response'))[''])")

585627: illegal character inside unquoted field at 7

("message":" File '\\var\\dataprophet\\omni-staging\\venv\\lib\\python3.6\\site-packages\\celery\\app\\builtins.py\\', line 83, in unlock_chord")

585629: illegal character inside unquoted field at 7

("message":" File '\\var\\dataprophet\\omni-staging\\venv\\lib\\python3.6\\site-packages\\celery\\result.py\\', line 765, in join")

585631: illegal character inside unquoted field at 7

("message":" File '\\var\\dataprophet\\omni-staging\\venv\\lib\\python3.6\\site-packages\\celery\\result.py\\', line 215, in get")

585633: illegal character inside unquoted field at 7

("message":" File '\\var\\dataprophet\\omni-staging\\venv\\lib\\python3.6\\site-packages\\celery\\result.py\\', line 221, in make_thread")

⚠ Some documents could not be imported

8 out of 696673 documents could not be imported. This could be due to lines not matching the Grok pattern.

▼ Failed documents

585631: illegal character inside unquoted field at 7

("message":" File '\\var\\dataprophet\\omni-staging\\venv\\lib\\python3.6\\site-packages\\celery\\result.py\\', line 215, in get")

585633: illegal character inside unquoted field at 7

("message":" File '\\var\\dataprophet\\omni-staging\\venv\\lib\\python3.6\\site-packages\\celery\\result.py\\', line 331, in maybe_throw")

585635: illegal character inside unquoted field at 7

("message":" File '\\var\\dataprophet\\omni-staging\\venv\\lib\\python3.6\\site-packages\\celery\\result.py\\', line 324, in throw")

585637: illegal character inside unquoted field at 7

("message":" File '\\var\\dataprophet\\omni-staging\\venv\\lib\\python3.6\\site-packages\\vine\\promises.py\\', line 244, in throw")

585639: illegal character inside unquoted field at 7

("message":" File '\\var\\dataprophet\\omni-staging\\venv\\lib\\python3.6\\site-packages\\vine\\five.py\\', line 195, in reraise")

Clean up

Ingest Node Pipelines

Define a pipeline for preprocessing documents before indexing.

<input type="checkbox"/> Name ↑
<input type="checkbox"/> omni-staging-access-pipeline
<input type="checkbox"/> omni-staging-error-pipeline
<input type="checkbox"/> staging-web-worker-pipeline
<input type="checkbox"/> staging-web-worker1-pipeline
<input type="checkbox"/> staging-worker-1-pipeline
<input type="checkbox"/> staging-worker-2-pipeline
<input type="checkbox"/> staging-worker-3-pipeline
<input type="checkbox"/> staging-worker-4-pipeline

Rows per page: 10 ▾

staging-worker-1-pipeline

Description

Ingest pipeline created by file structure finder

Processors

```
[
  {
    "dissect": {
      "field": "message",
      "pattern": "[%{timestamp}, %{pid}: %{loglevel}/%{worker}]"
    }
  }
]
```

Ingest Node Pipelines

Define a pipeline for preprocessing documents before indexing.

<input type="checkbox"/> Name ↑	Actions
<input type="checkbox"/> omni-staging-access-pipeline	...
<input type="checkbox"/> omni-staging-error-pipeline	...
<input type="checkbox"/> omni-web-access-pipeline	...
<input type="checkbox"/> omni-web-test-access-dissect-pipeline	...
<input type="checkbox"/> omni-web-test-error-dissect-pipeline	...
<input type="checkbox"/> omniweb-pipeline	...
<input type="checkbox"/> omniweb_access_logs-pipeline	...
<input type="checkbox"/> xpack_monitoring_6	...

Rows per page: 10 ▾

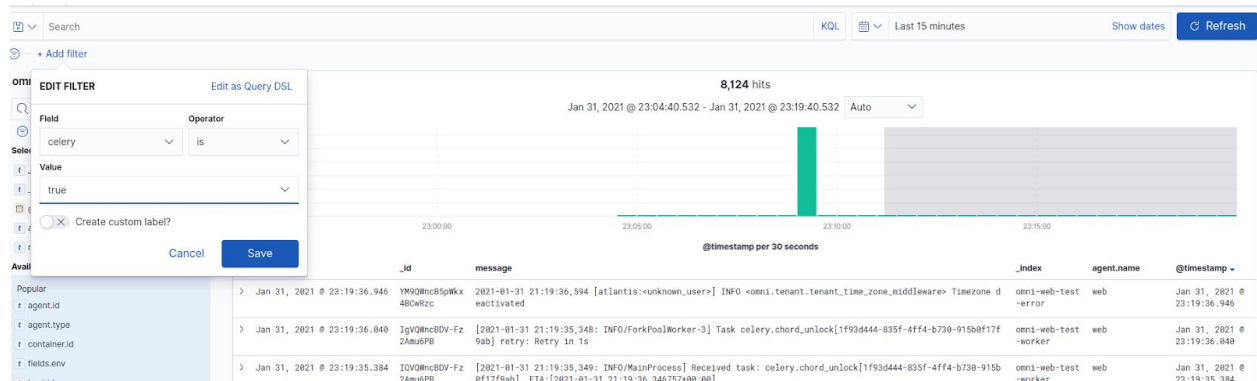
< 1 >

[Ingest Node Pipelines docs](#)

Grok pattern:

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

Testing Staging Workers with live logs



Test all test case

Question

The screenshot shows the 'Create pipeline' form in Kibana. The form has fields for 'Name', 'Description', 'Processors', and 'Failure processors'. The 'Name' field is filled with 'staging-web-worker-pipeline-copy'. The 'Description' field is filled with 'Ingest pipeline created by file structure'. The 'Processors' section shows a 'Grok' processor. The 'Test pipeline' sidebar on the right shows the 'Documents' tab with an error message: '[security_exception] action [indices/data/read/get] is unauthorized for user [tshepiso.moloko]'. The 'Index' field is filled with 'omni-web-test-*' and the 'Document ID' field is filled with 'XQVSWncBDV-Fz2AmF6mA'.

Testing grok pattern with the new logs

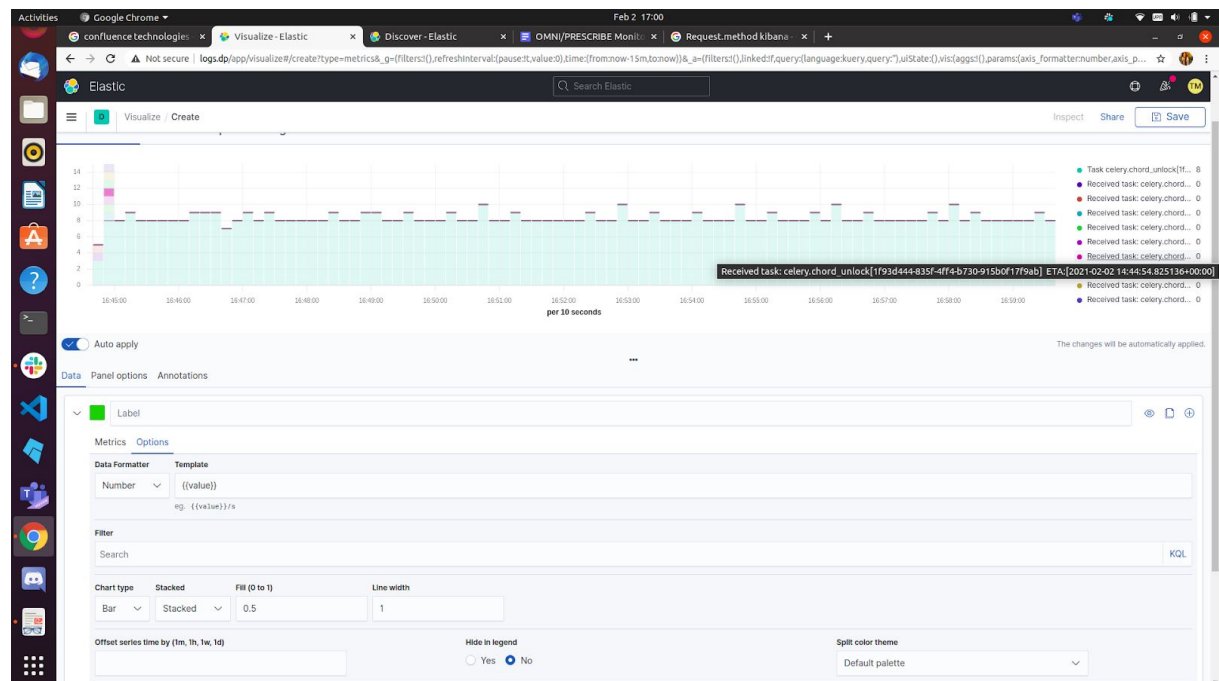
DASHBOARDING - what to look for in monitoring dashboard distributed web applications

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.

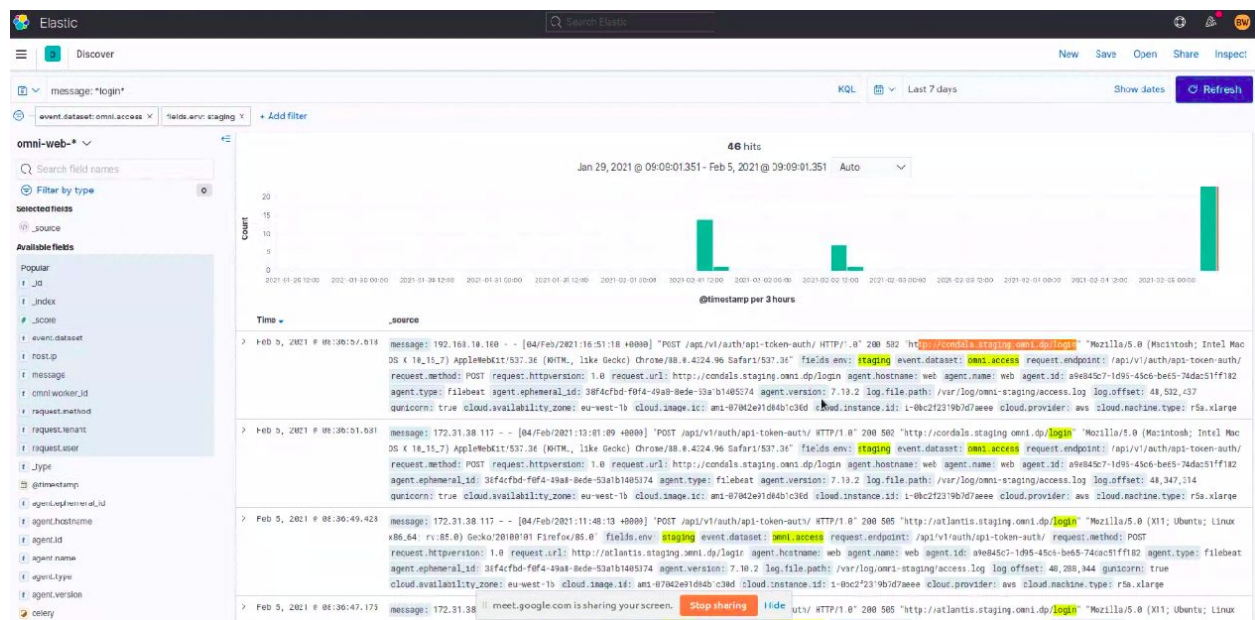

```
2021-02-08T07:25:58.757Z DEBUG [elasticsearch] elasticsearch/client.go:414 Bulk item insert failed (i=0, status=500): {"type":"find_match","reason":"Unable to find match for dissect pattern: [%
(event.created)] [%{process.pid}] [%{log.level}] [%{message}] against source: /var/dataprophet/omni-staging/venv/lib/python3.6/site-packages/django/db/models/fields/__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})"
```

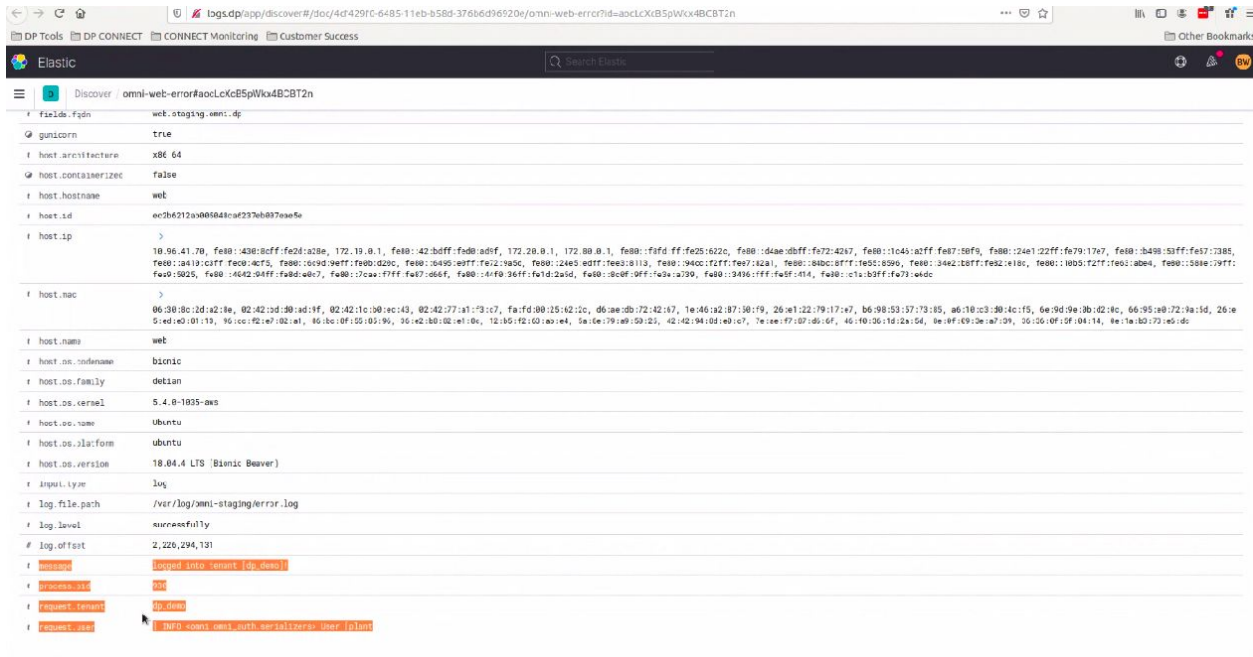
- When sources are broken - sudo conditions to fix this error
- Not treating that as multilines

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] 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)s <%(name)s> %(message)s"
```

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