DRAFT BLOG POST

	Content
Post Title	Exploring an ELK Stack: Part 1: Importing Data and Patterns
Post Date	
Attributed To	Peter Welcher
Written By	Peter Welcher
Reviewed By (Name & Date)	Dave Donati (10/29)
Reviewed By (Name & Date)	

Meta Title (55 characters including spaces)	Exploring an ELK Stack: Part 1: Importing Data and Patterns
Meta Description (156 characters including spaces)	
Target Keywords	
Categories	Technology
Tags	N/A
Call to Action	N/A
Image	Put a brief description of what the image should be or note that a file is attached. DO NOT paste the image into this Word doc; send it as a separate file.

COPY FOR POST:

This blog is the first of a set of getting-started tutorial blogs with screen captures, written as I explore the ELK stack and related tools. I hope that as such a tutorial, this is on par with or somewhat more detailed than what I've found on the web, and with more of a network focus.

As a tutorial with screen captures, this will run longer than most blogs. Hopefully not "TMI".

The main purpose here is to introduce the ELK stack, and show some of what it can do in the context of an actual network application.

What is ELK?

ELK stands for ElasticSearch, LogStash, Kibana. An ELK stack is common in the VM and container worlds, and often used to provide dashboards for key metrics like app cluster load, response time, etc. I feel the need to include words like “scale-out”, “clusters”, and “cloud-native” here.

Our explorations will be more focused on the capabilities side: ***Why would I want to use an ELK stack?*** What can it do for me?

Here’s the short summary of what the ELK components do:

- LogStash handles import of log data
- ElasticSearch provides storage and lookups
- Kibana provides user front end for searching the data, visualizing (graphing counts and other metrics in various ways), and dashboards.

Another tool which can be useful with an ELK stack: Beats agents (e.g. on servers, etc.) can transform and forward various types of data to ElasticSearch.

I claim no great depth of knowledge here (still learning), but I am hoping to ease your explorations in this area, should you choose to do so. I’ll provide details where I deem it useful: sometimes online instructions either inadvertently omitted a step (or I missed the “obvious”), or things changed since the instructions were prepared, which will no doubt happen to this blog as well.

Caution

I should note that building your own robust ELK cluster, and how much configuration and coding to do with it, are both matters to consider. You can purchase ELK as a service, and you can also purchase toolkits and consulting services to get you up and running faster. Time (to build and support) versus money — an analogy might be using a configuration-heavy management tool that is free or low cost, versus spending a lot on a tool that does more for you automatically or more easily.

Getting Started

I work on a Mac. I’ve got an Ubuntu VM running in Fusion to (a) simplify installation of open source packages (no Mac quirks!), and (b) help me control disk space, namely the install bloat from downloading and exploring different Linux packages, VMs, and containers. It’s the twisty little trails of dependencies that get installed … I can just go back to a clean Ubuntu VM image, as opposed to having to hunt down and uninstall stuff.

So, my instructions below are going to be for a Ubuntu system.

I’ve been working with an ELK stack container, since that isolates the ELK experimentation from the rest of the Ubuntu system, and requires minimal effort to

get rolling with. The main drawback is if you want to alter files within the container, it takes more effort.

To use containers, the starting point is to get Docker installed. See the *References* below for guidance.

Once you've done that, the next step is very difficult (joking):

```
docker pull sep/elk
```

That pulls from https://hub.docker.com/r/sebp/elk

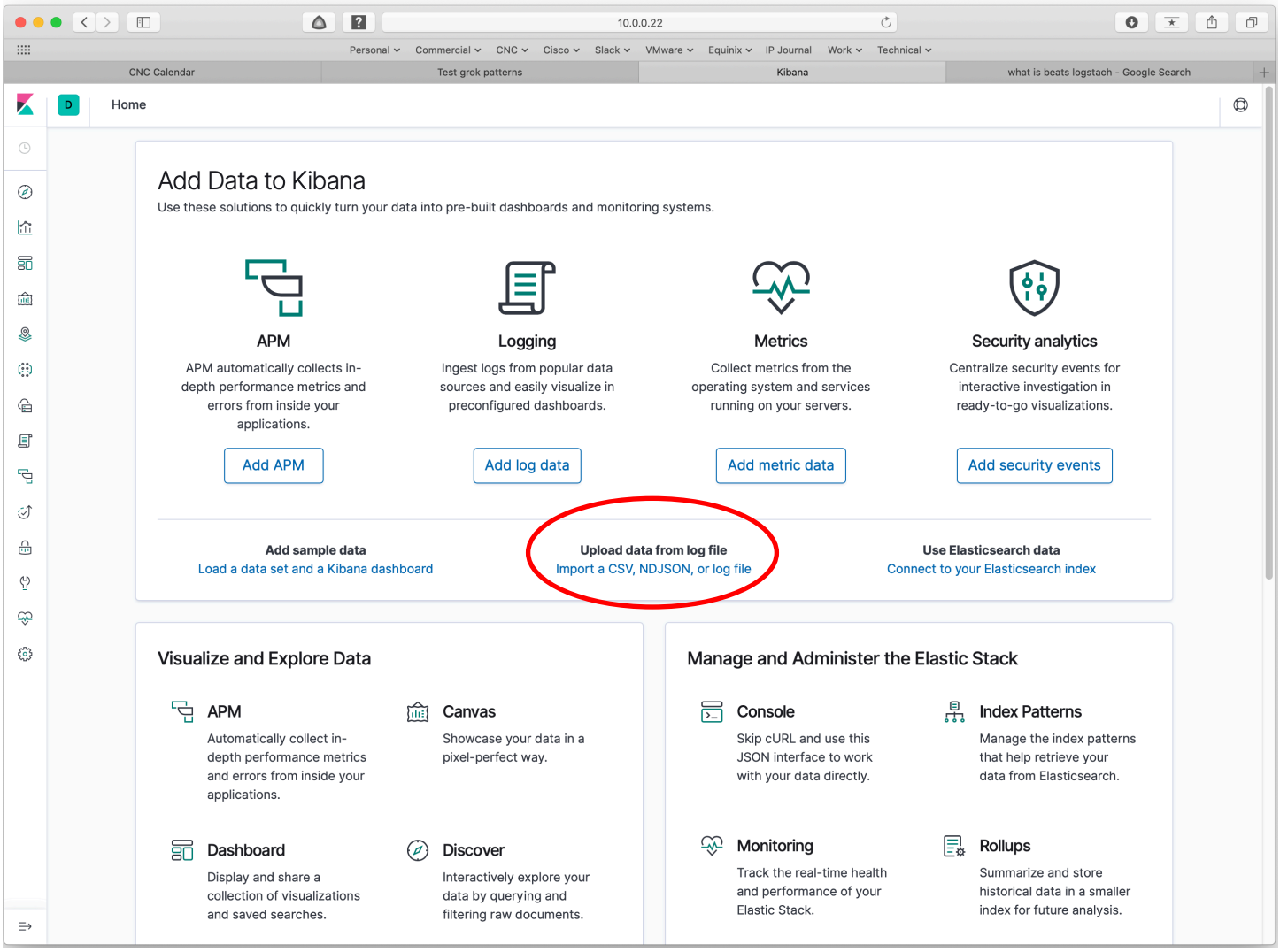
I then run it with this bash script:

```
#! /bin/bash
# Fix one system setting
sudo sysctl -w vm.max_map_count=280000

sudo docker run \
-p 5601:5601 -p 9200:9200 -p 5044:5044 \
-e "discovery.type=single-node" \
-it --name elk sebp/elk

echo "port 5601 (Kibana web interface), 9200 (Elasticsearch JSON
interface) 5044 (Logstash Beats)"
```

The last part is a reminder of the ports used.

The run stays associated with the terminal window, where you'll see logging output. You'll need to be patient; it takes a few tens of seconds for things to start up. Use Control-C when done to easily kill the container off.

Once the various services are spun up, you can then web into them. Use http://localhost:9200 to make sure ElasticSearch is running OK. You can then use http://localhost:5601 to hit the Kibana home page, shown below.



What Now?

What do we DO with this now?

When I see the word “log” I immediately think “syslog”. I’m a big fan of central syslog data collection and have been fiddling with a PERL script to post-process the data for years. Timestamps and general formatting vary between sites.

The simplest source of information would be to configure network devices to send to the container. Unfortunately, I disposed of my home lab in favor of VIRL, and then stopped using it locally because a company VM can provide a lot more RAM for bigger models. Since that’s remote, exporting the data back to my Mac would not be … simple. My goal was to explore basic Kibana log analysis, so how to avoid getting bogged down?

I have captured syslog data from various consulting work, so why not use some of that? I tried some things that didn't work out well and gave up a couple of months back: too much time consumed. Revisiting it more recently, I found the following.

A New Feature to the Rescue!

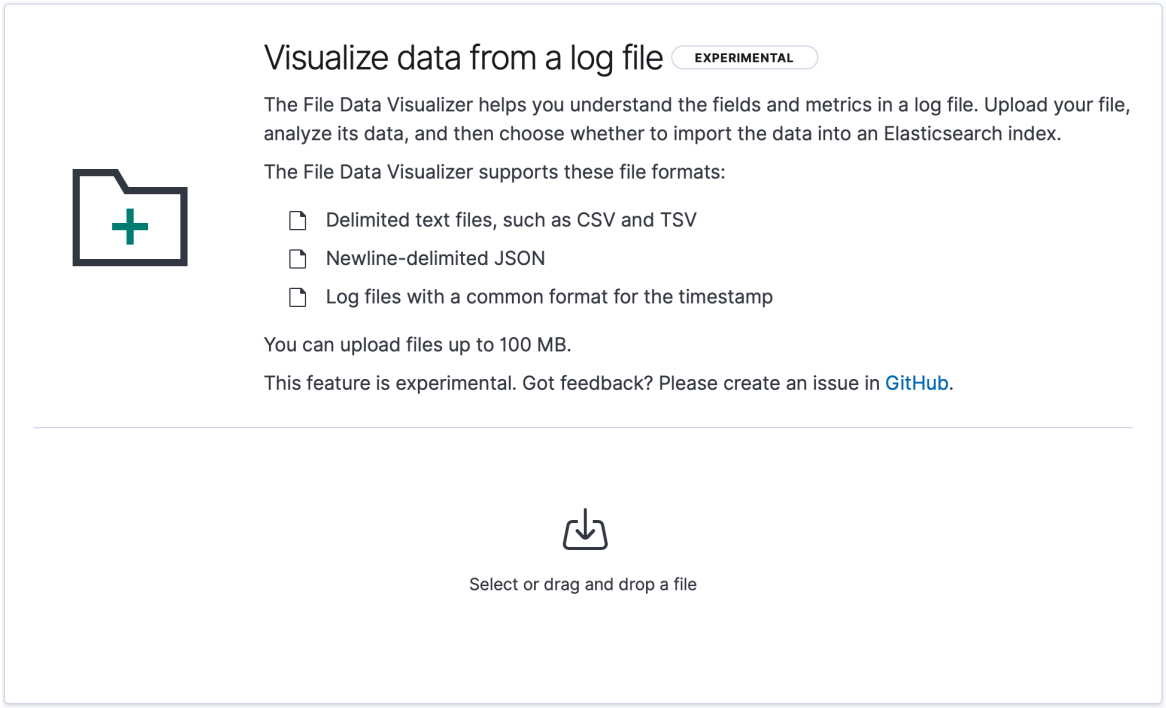
Thanks to the blog https://www.elastic.co/blog/importing-csv-and-log-data-into-elasticsearch-with-file-data-visualizer, which got me started in the right direction.

It turns out there is a new experimental feature in ElasticSearch / Kibana: manual import of log data. The following screen captures are tightly focused on syslog data, whereas the above blog was more focused on CSV data import.

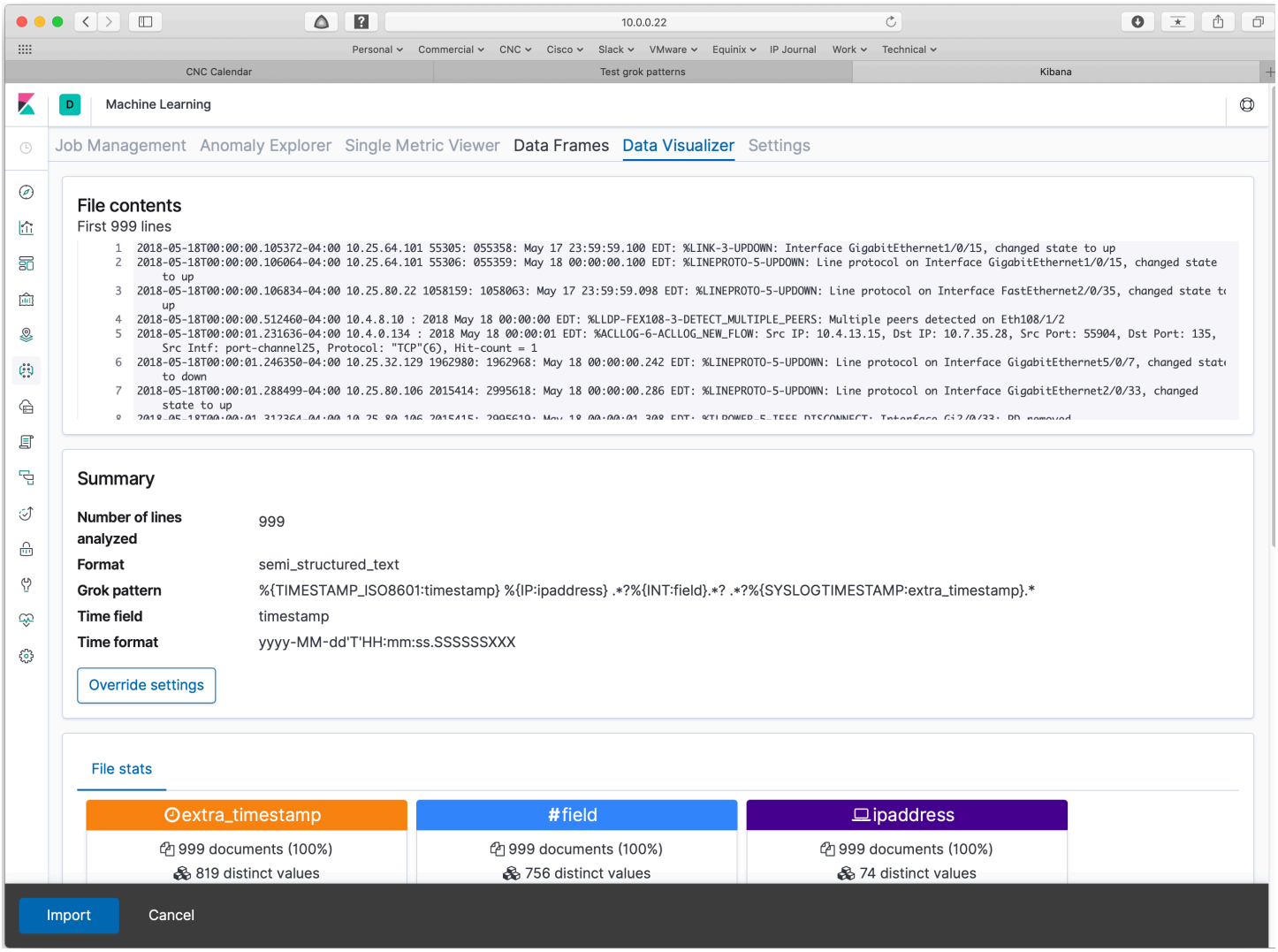
It notes that the size of the file that is imported is limited to 100 MB. So, this feature is handy for spot analysis and learning right now, but not for full syslog from many Cisco devices (I was going to word that as "network devices", but I am greatly underwhelmed by the usefulness of Juniper and HP syslog messages).

That last caveat only refers to direct import from a file. Forwarding syslog to ELK is not size limited (other than by the size of your server, associated storage, etc., of course).

Note that the previous screen capture has a red circle around it. Clicking on the link in the circle brings the following screen up:

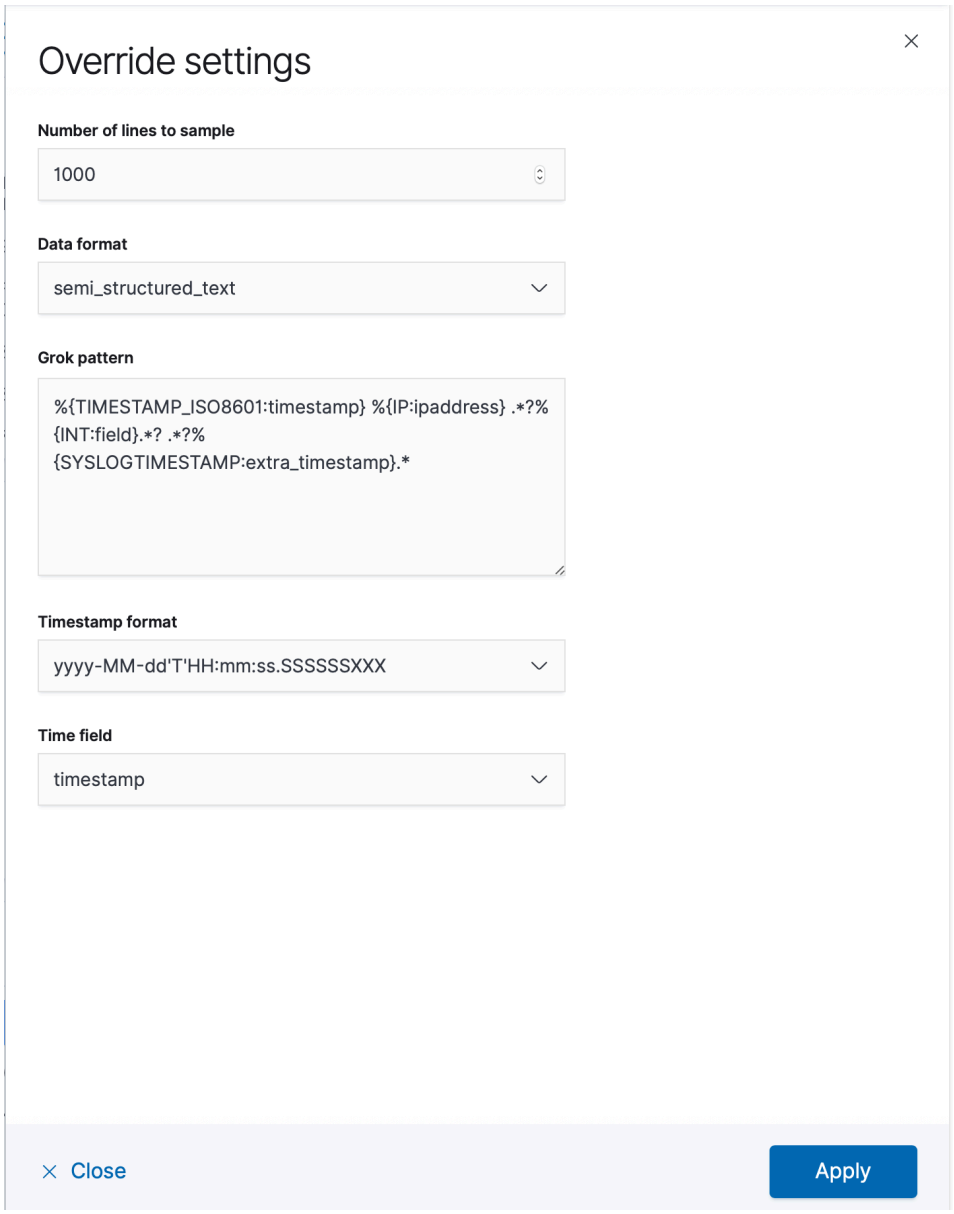


Click on the “Select” at the bottom to start log data import. Choose a file of syslog data.



The system tries to find a pattern matching the data. In this case, I’d like to do better. One of my objectives was to see whether this approach could potentially do better than the PERL script fiddling I’ve been doing.

Clicking on “Override settings” in the above lets you supply your own pattern. It brings up:



After a good bit of experimenting, I replaced the pattern with the line:

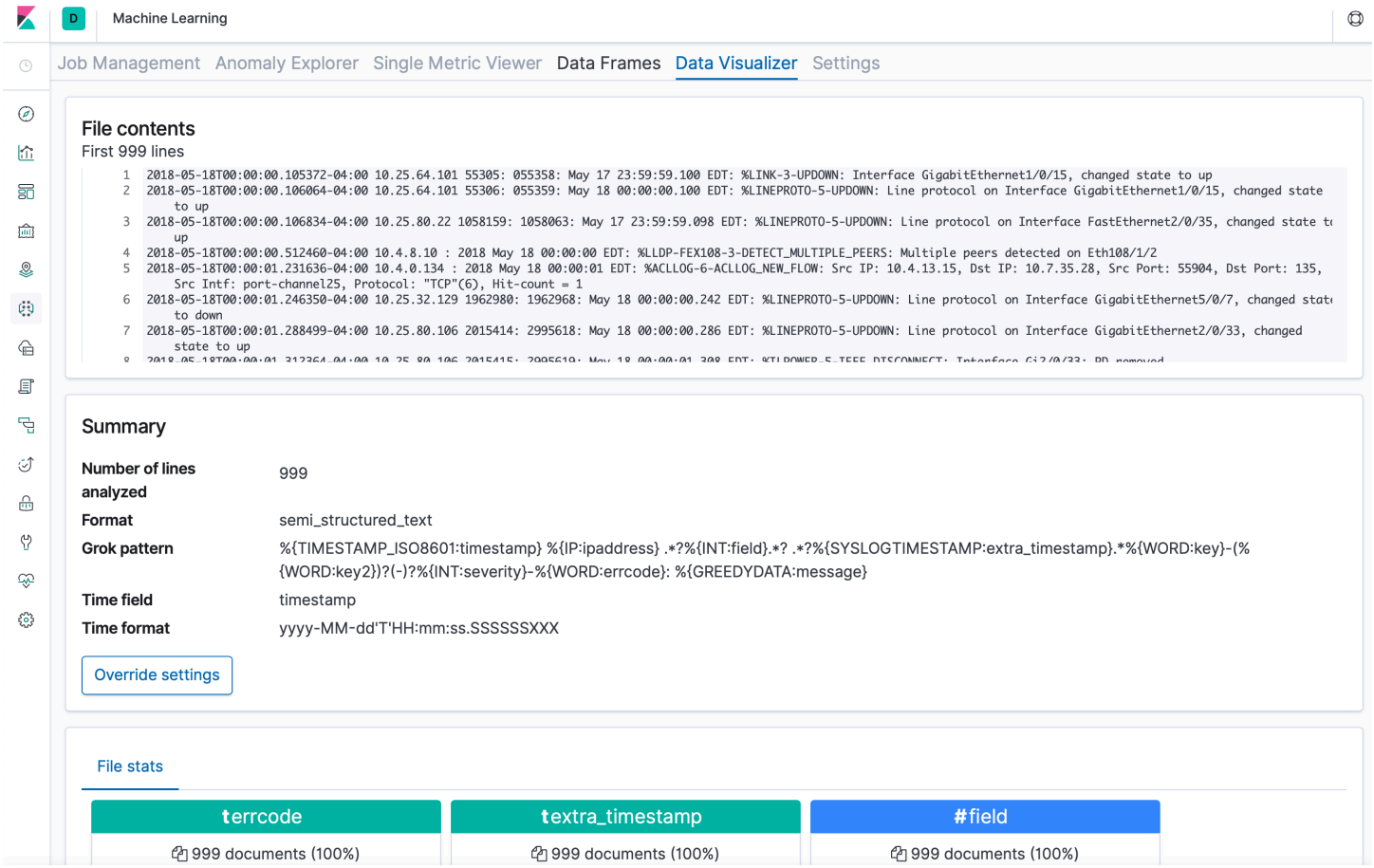
```
%{TIMESTAMP_ISO8601:timestamp} %{IP:ipaddress} .*?%{INT:field}.*?
.*?%{SYSLOGTIMESTAMP:extra_timestamp}.*%{WORD:key}-(%{WORD:key2})?(-
)?%{INT:severity}-%{WORD:errcode}: %{GREEDYDATA:message}
```

And clicked Apply.

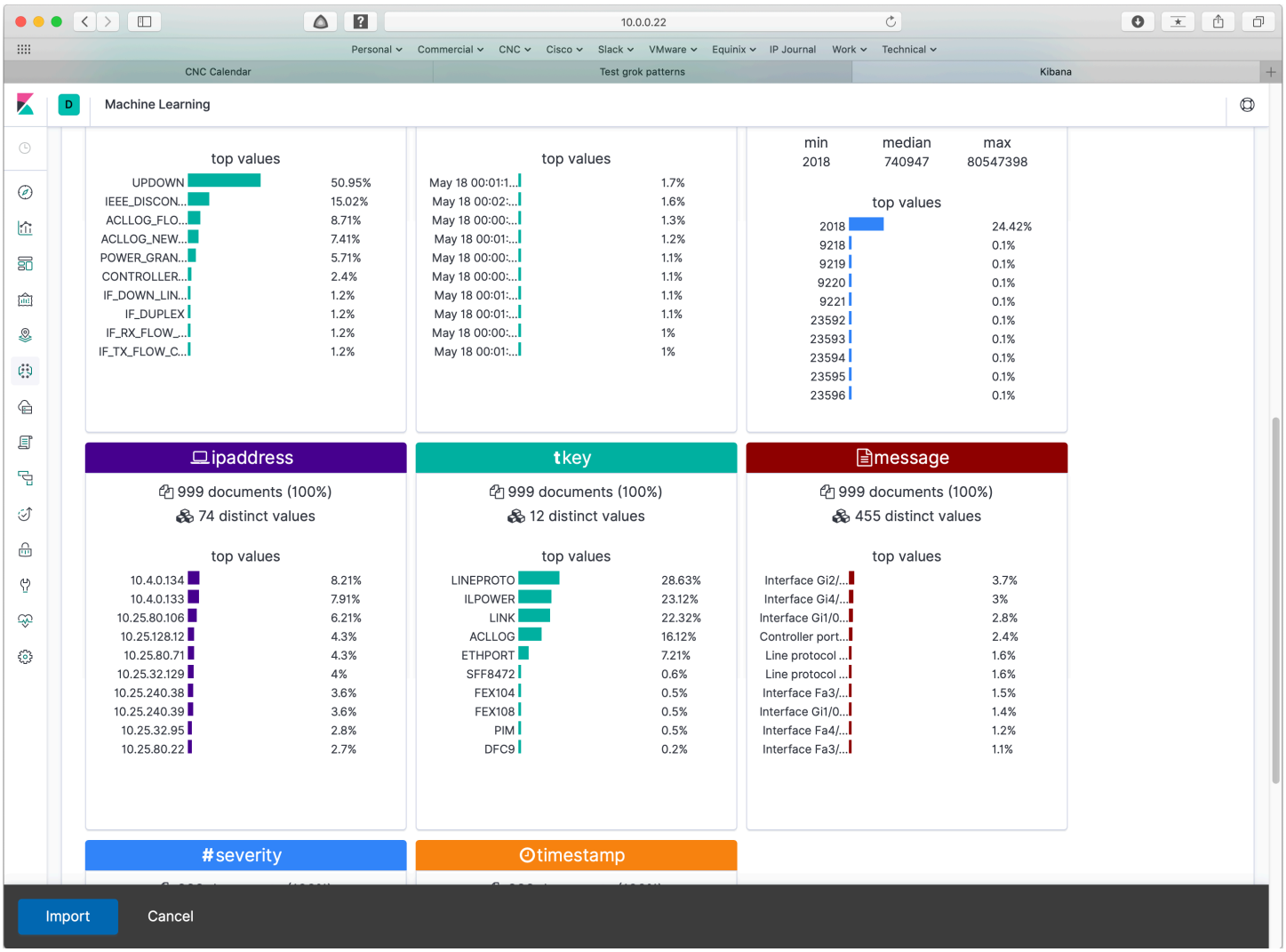
Note: Your Mileage May Vary. I've seen a lot of different formats for storing syslog data over the years. The above may well need to be adjusted to better fit your site's format.

"Experimenting" here means trying a pattern, clicking Apply and other steps, then starting over. More on that below: we'll come back to patterns. There's a better way to do that.

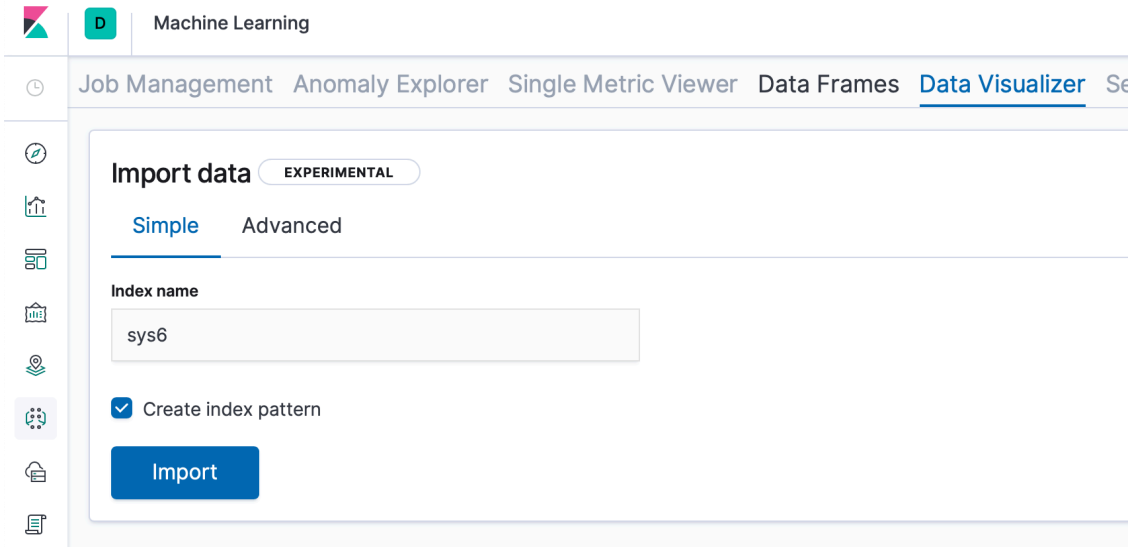
That brings up the following:



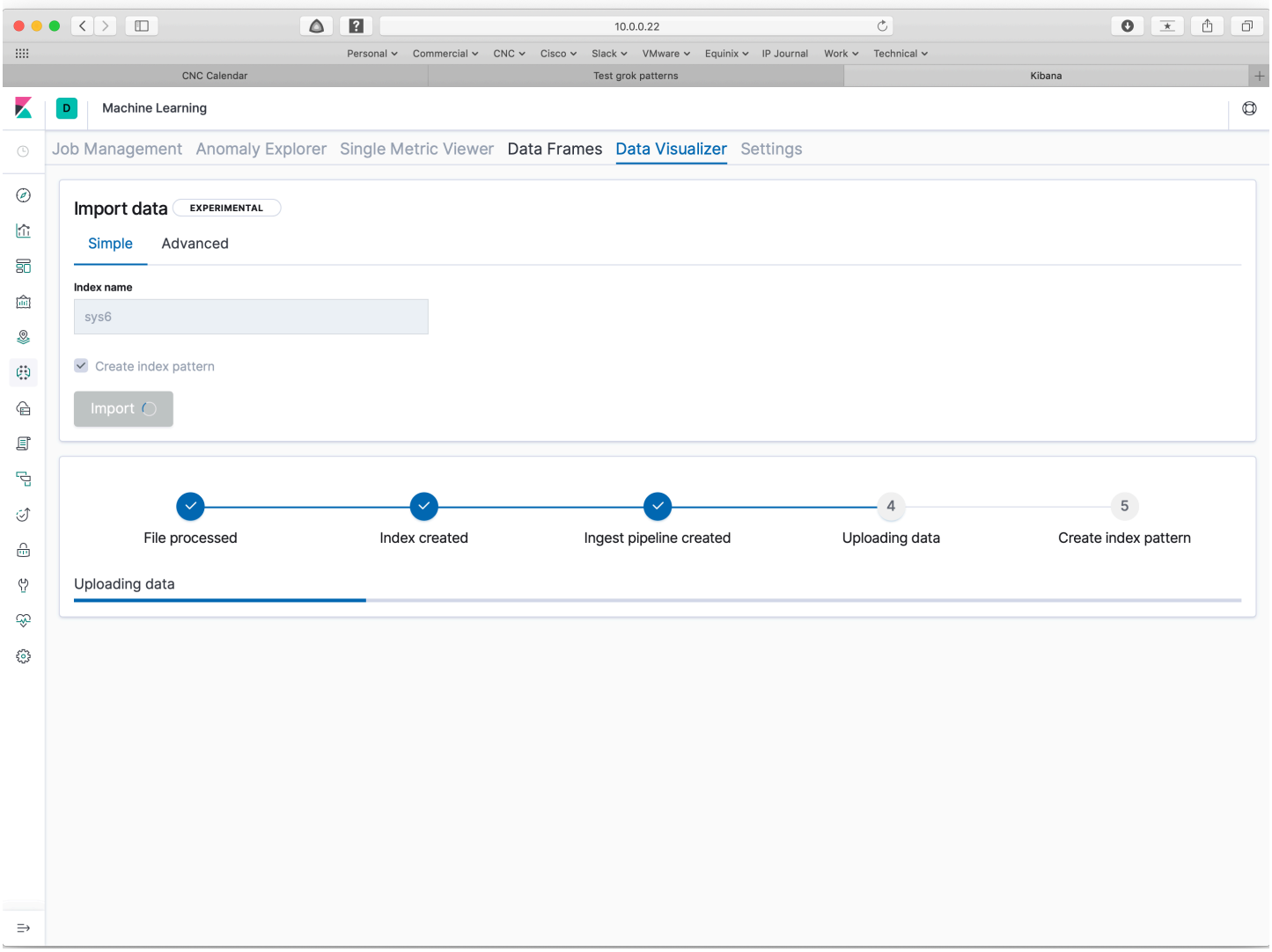
Scrolling a bit confirms the data is being read properly (note the values listed for each of the fields):



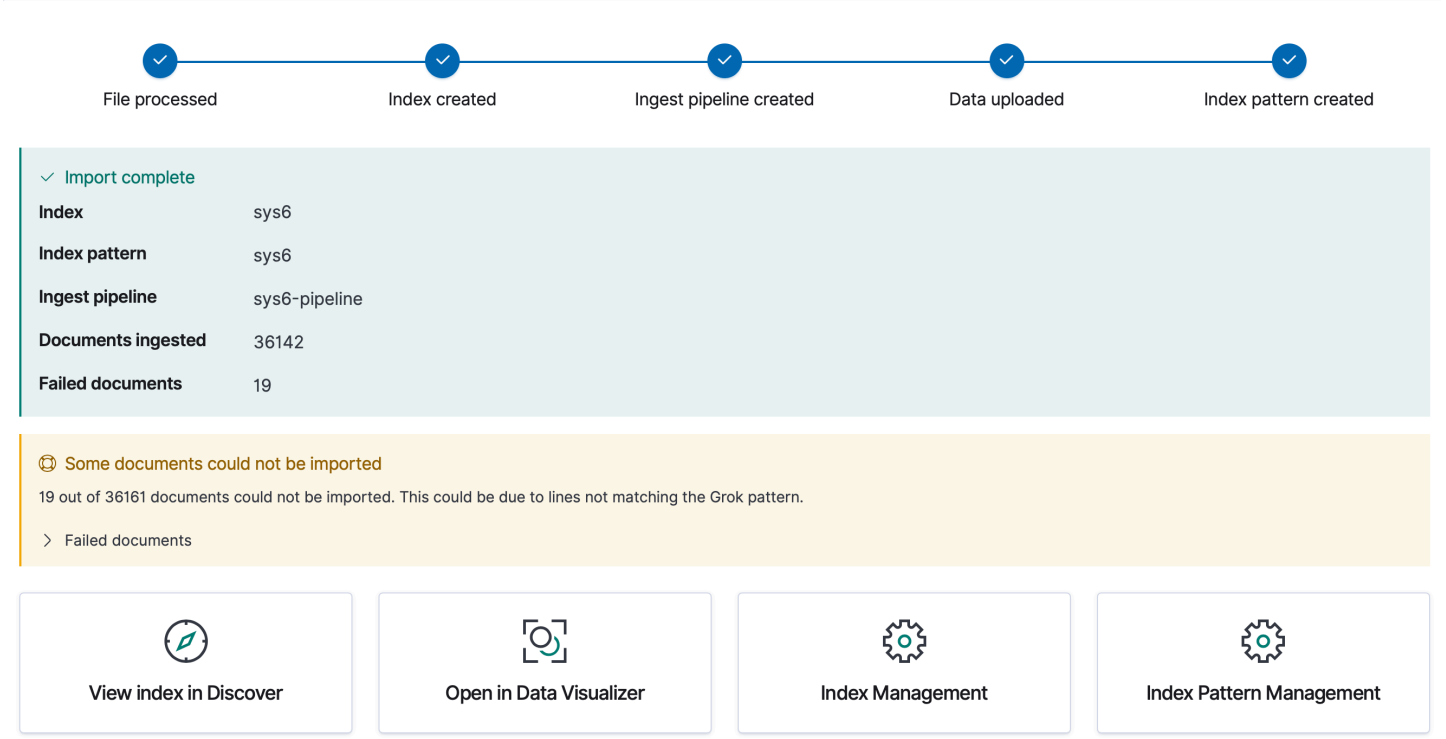
Clicking on Import then prepares to import the data, processing it according to the pattern. That brings up the next screen, shown below. You do then have to supply a name for the data and pattern.



Click on Import, and you will see status:



When that finishes you get a status summary at the bottom of that screen / page:



That indicates that 36142 lines were read, and 19 failed to match the pattern. Not bad!

You won't be able to proceed if the initial syslog lines don't match the pattern.

For received data processed by LogStash, I believe that lines that don't match are just ignored, just as occurred above.

Grok Patterns

From the above, you can see that the patterns are rather important in getting our syslog data into the system.

Google search led to some hits, listed under References below, but neither seemed aligned with where I wanted to go. I also had some problems with the match patterns, but I appreciate them since they helped me get started.

Let's discuss grok patterns.

The basic way data is handled is to apply "grok" filters to inbound data to provide structure to that data, including field names. The filters use patterns for that. There is a file in the ELK container with the standard provided patterns.

You can see what people have built at https://github.com/logstash-plugins/logstash-patterns-core/blob/master/patterns/grok-patterns. They are just named regular expression patterns.

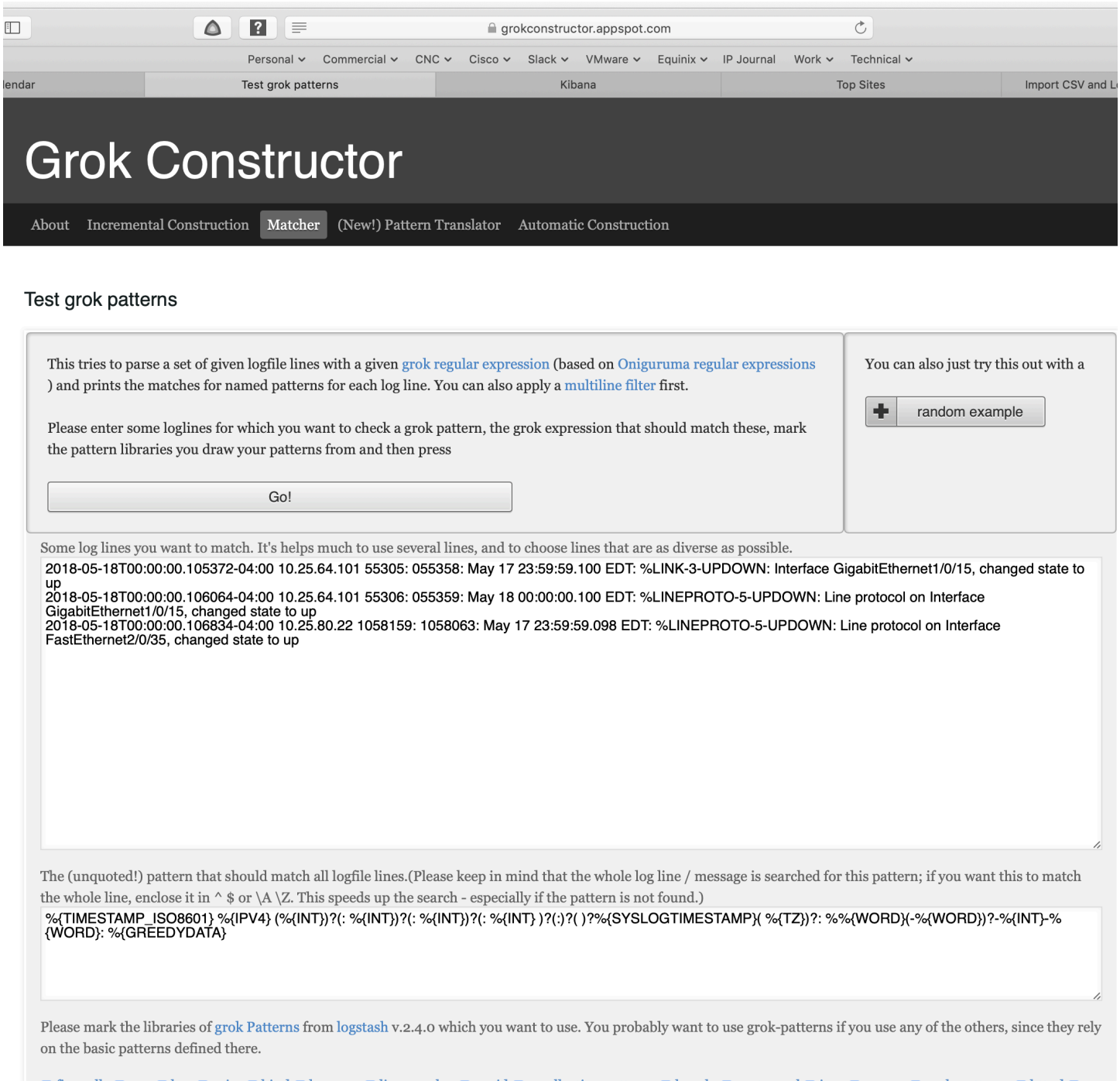
Example:

```
USERNAME [a-zA-Z0-9._-]+
```

You would reference that as %{USERNAME} in an ELK pattern.

I started down the path of some complex and ugly regular expressions, then discovered two very useful tools: Grok Constructor and Grok Debugger.

Here's a screen capture from Grok Constructor (you can tell that because it says so):



You put sample syslog in the top dialog box (it will accept a LOT of lines – start small, then check against a big sample once you think you've got the pattern right). Put your pattern in the pattern box, click "Go!", and below that it'll tell you what matched and didn't match the pattern.

That very flexible regex gave Kibana indigestion and may have been overkill, but it matched a lot of lines.

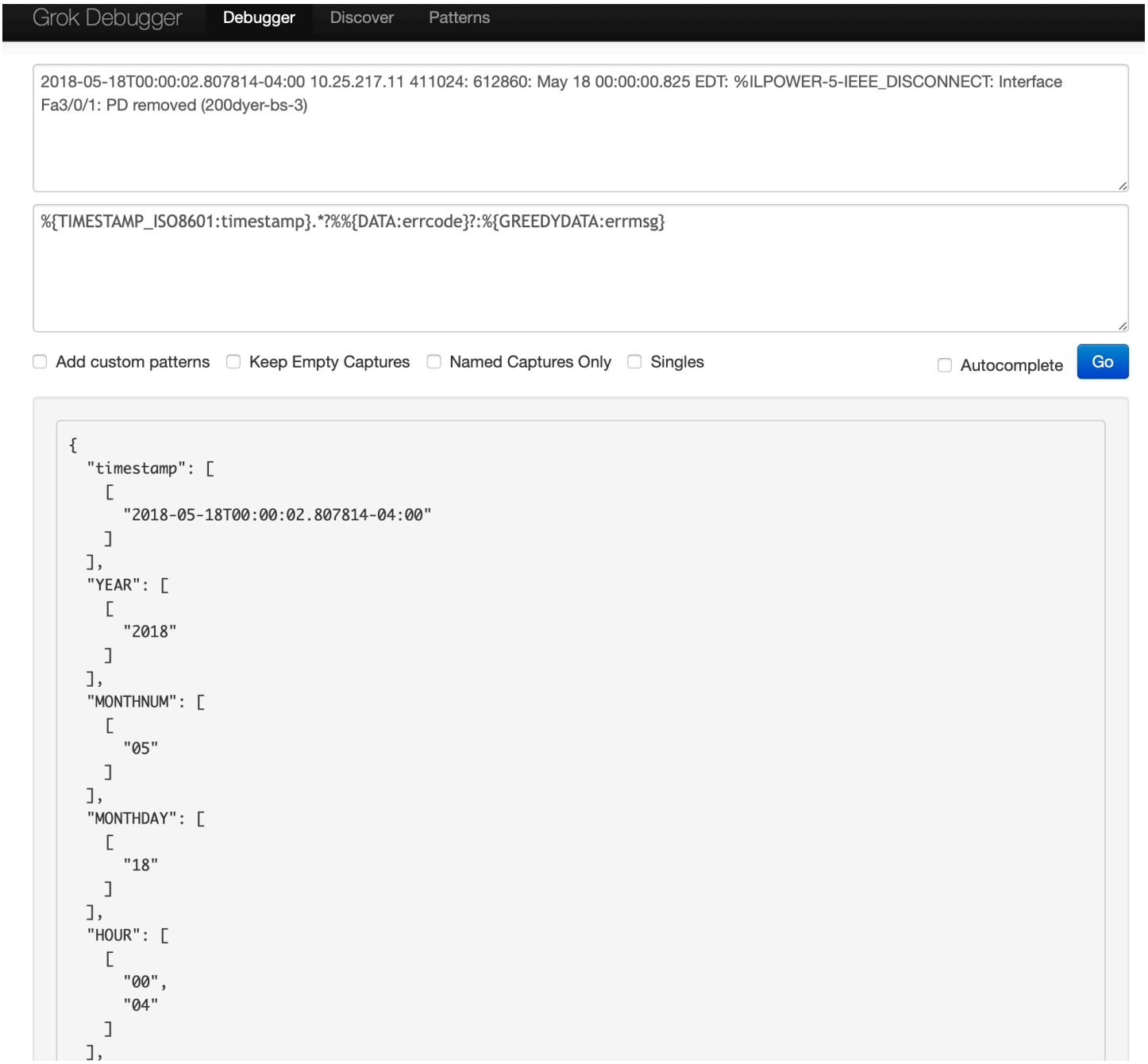
Hint: putting parentheses and ? around a grok pattern makes it optional.

Where this was useful was (a) trying to get the fields pulled out cleanly by Kibana, and (b) trying to figure out which inscrutable lines to remove from my syslog sample:

```
pjw$ grep -v 'last message repeated' sample | grep -v 'RP/0' | grep -v
'aruba' | grep -v 'TACACS' >sample2
```

For purposes of learning, there seemed little point in trying to digest every syslog message format I had in the sample. The above eliminated the odder ones, getting the non-matches down to 19 out of about 36000 lines using the much simpler pattern shown earlier in this blog.

Here's a screen capture from Grok Debugger:



You put one or more lines in the top, pattern below that, hit “Go” and see what matched. It’s a bit odd in that subsequent matches against any type (like INT) all show up in a list in one place — I was expecting sequential aligned with the pattern components.

Conclusion

The above got our syslog data into the Elasticsearch search engine ready to be analyzed. Since this blog is already rather long, I've deferred the analysis / visualization part for a second blog (Tech blog "cliff-hanger" achieved!).

Murphy's Law says that once I got the above working, I'd find another way. See the 3rd page (or thereabouts) in the logz.io Kibana tutorial for a way to load data via LogStash. Note that if you're using containers, you'll either have to copy the data into the container or mount a file / folder on the local disk as a volume to make the data accessible within the container.

The approach described here has the virtue of uploading via the web browser. Simpler?

By the way, you're likely a science fiction fan (and perhaps old) if you know where "grok" comes from. For myself, I'll concede the fan part, but in denial on the old part.

References

- Install Docker Engine: https://docs.docker.com/install/
- Install Docker Enginer Ubuntu: https://docs.docker.com/install/linux/docker-ce/ubuntu/
- Grok Constructor: https://grokconstructor.appspot.com/do/match
- Grok Debugger: https://grokdebug.herokuapp.com
- Grokking "grok": https://www.merriam-webster.com/dictionary/grok
- Logz.io Kibana Tutorial (loading data in via Logstash): https://logz.io/blog/kibana-tutorial/

Google search hits ("cisco syslog elasticsearch"):

- https://www.neteye-blog.com/2017/10/sending-cisco-syslogs-to-elasticsearch-a-simple-guide/
- https://gist.github.com/justinjahn/85305bc7b7df9a6412baedce5f1a0ece

Comments

Comments are welcome, both in agreement or constructive disagreement about the above. I enjoy hearing from readers and carrying on deeper discussion via comments. Thanks in advance!

Hashtags: #CiscoChampion #TechFieldDay #TheNetCraftsmenWay

Twitter: @pjwelcher

Disclosure Statement

[INSERT the usual IMAGES HERE: 20 Year CCIE and Cisco Champions **2019** as per recent blogs]

NetCraftsmen Services

Did you know that NetCraftsmen does network /datacenter / security / collaboration design / design review? Or that we have deep UC&C experts on staff, including @ucguerilla? For more information, contact us at <<insert suitable link here>>.

Social Media:

Facebook: Like, comment or share our status using this link.

Twitter: Like and RT our tweet using this link.

LinkedIn: Like, comment or share our status using this link.