



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	<p>Put a brief description of what the image should be or note that a file is attached.</p> <p><u>DO NOT</u> paste the image into this Word doc; send it as a separate file.</p>

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 sebp/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.





The screenshot shows the Kibana interface with a sidebar containing various icons. The main area is titled 'Add Data to Kibana' and contains four sections: APM, Logging, Metrics, and Security analytics. Below these are three buttons: 'Add sample data', 'Upload data from log file' (which is circled in red), and 'Use Elasticsearch data'. The 'Upload data from log file' section includes a sub-instruction 'Import a CSV, NDJSON, or log file'. At the bottom, there are two larger sections: 'Visualize and Explore Data' and 'Manage and Administer the Elastic Stack', each with several sub-options.

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:

Visualize data from a log file EXPERIMENTAL

The File Data Visualizer helps you understand the fields and metrics in a log file. Upload your file, analyze its data, and then choose whether to import the data into an Elasticsearch index.

The File Data Visualizer supports these file formats:



- Delimited text files, such as CSV and TSV
- Newline-delimited JSON
- Log files with a common format for the timestamp

You can upload files up to 100 MB.

This feature is experimental. Got feedback? Please create an issue in [GitHub](#).

Select or drag and drop a file







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

The screenshot shows the NetCraftsmen Data Visualizer interface. At the top, there's a navigation bar with links like Personal, Commercial, CNC, Cisco, Slack, VMware, Equinix, IP Journal, Work, and Technical. Below the navigation bar, there are tabs for CNC Calendar, Test grok patterns, and Kibana. The main area has a sidebar with various icons and a title "Machine Learning". Below the sidebar, there are several sections: "File contents" showing the first 999 lines of a log file with some redacted text; "Summary" with details like Number of lines analyzed (999), Format (semi_structured_text), Grok pattern (%{TIMESTAMP_ISO8601:timestamp} %{IP:ipaddress}.*%{INT:field}.*.*?%{SYSLOGTIMESTAMP:extra_timestamp}.*), Time field (timestamp), and Time format (yyyy-MM-dd'T'HH:mm:ss.SSSSSSSXX); "Override settings" button; and "File stats" section with three cards: "extra_timestamp" (999 documents, 100%, 819 distinct values), "#field" (999 documents, 100%, 756 distinct values), and "ipaddress" (999 documents, 100%, 74 distinct values). At the bottom, there are "Import" and "Cancel" buttons.

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:





Override settings

Number of lines to sample
1000

Data format
semi_structured_text

Grok pattern

```
%{TIMESTAMP_ISO8601:timestamp} %{IP:ipaddress} .*%{INT:field}.*?%{SYSLOGTIMESTAMP:extra_timestamp}.*
```

Timestamp format
yyyy-MM-dd'T'HH:mm:ss.SSSSSXXX

Time field
timestamp

[Close](#) [Apply](#)

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:

The screenshot shows the Data Visualizer interface with the following sections:

- File contents:** Displays the first 999 lines of the log file. Lines 1 through 8 are shown as examples:
 - 2018-05-18T00:00:00.105372-04:00 10.25.64.101 55305: 055358: May 17 23:59:59.100 EDT: %LINK-3-UPDOWN: Interface GigabitEthernet1/0/15, changed state to up
 - 2018-05-18T00:00:00.106064-04:00 10.25.64.101 55306: 055359: May 18 00:00:00.100 EDT: %LINEPROTO-5-UPDOWN: Line protocol on Interface GigabitEthernet1/0/15, changed state to up
 - 2018-05-18T00:00:00.106834-04:00 10.25.80.22 1058159: 1058063: May 17 23:59:59.098 EDT: %LINEPROTO-5-UPDOWN: Line protocol on Interface FastEthernet2/0/35, changed state to up
 - 2018-05-18T00:00:00.512460-04:00 10.4.8.10 : 2018 May 18 00:00:00 EDT: %LLDP-FEX108-3-DETECT_MULTIPLE_PEERS: Multiple peers detected on Eth108/1/2
 - 2018-05-18T00:00:01.231636-04:00 10.4.0.134 : 2018 May 18 00:00:01 EDT: %ACLLOG-6-ACLLOG_NEW_FLOW: Src IP: 10.4.13.15, Dst IP: 10.7.35.28, Src Port: 55904, Dst Port: 135, Src Intf: port-channel125, Protocol: "TCP"(6), Hit-count = 1
 - 2018-05-18T00:00:01.246350-04:00 10.25.32.129 1962980: 1962968: May 18 00:00:00.242 EDT: %LINEPROTO-5-UPDOWN: Line protocol on Interface GigabitEthernet5/0/7, changed state to down
 - 2018-05-18T00:00:01.288499-04:00 10.25.80.106 2015414: 2995618: May 18 00:00:00.286 EDT: %LINEPROTO-5-UPDOWN: Line protocol on Interface GigabitEthernet2/0/33, changed state to up
 - 2018-05-18T00:00:01.212361-04:00 10.25.80.106 2005610: May 18 00:00:01.308 EDT: %TI-DWDM_E-TREE_DTSCONNECT: Transistor Gi2/0/23, DN removed
- Summary:** Provides an overview of the analyzed data:
 - Number of lines analyzed: 999
 - Format: semi_structured_text
 - Grok pattern: %{TIMESTAMP_ISO8601:timestamp} %{IP:ipaddress}.*?%{INT:field}.*? .*?%{SYSLOGTIMESTAMP:extra_timestamp}.*?%{WORD:key}-(%{WORD:key2})?-(-)?%{INT:severity}-%{WORD:errcode}: %{GREEDYDATA:message}
 - Time field: timestamp
 - Time format: yyyy-MM-dd'T'HH:mm:ss.SSSSSXXX
- File stats:** Shows the distribution of fields across 999 documents (100%):

terrcode	textra_timestamp	#field
999 documents (100%)	999 documents (100%)	999 documents (100%)

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





The screenshot shows the Kibana interface with several visualizations:

- Machine Learning** section:
 - Top values for **UPDOWN**: UPDOWN (50.95%), IEEE_DISCON... (15.02%), ACLLOG_FLO... (8.71%), ACLLOG_NEW... (7.41%), POWER_GRAN... (5.71%), CONTROLLER... (2.4%), IF_DOWN_LIN... (1.2%), IF_DUPLEX (1.2%), IF_RX_FLOW... (1.2%), IF_TX_FLOW_C... (1.2%).
 - Top values for **May 18 00:01:1...**: 1.7% (May 18 00:02:1...), 1.6% (May 18 00:00:1...), 1.3% (May 18 00:01:1...), 1.2% (May 18 00:00:1...), 1.1% (May 18 00:00:1...), 1.1% (May 18 00:01:1...), 1.1% (May 18 00:01:1...), 1% (May 18 00:00:1...), 1% (May 18 00:01:1...).
 - Min, median, max for **2018**: min 2018, median 740947, max 80547398.
- ipaddress** visualization: 999 documents (100%), 74 distinct values. Top values:
 - 10.4.0.134 (8.21%)
 - 10.4.0.133 (7.91%)
 - 10.25.80.106 (6.21%)
 - 10.25.128.12 (4.3%)
 - 10.25.80.71 (4.3%)
 - 10.25.32.129 (4%)
 - 10.25.240.38 (3.6%)
 - 10.25.240.39 (3.6%)
 - 10.25.32.95 (2.8%)
 - 10.25.80.22 (2.7%)
- tkey** visualization: 999 documents (100%), 12 distinct values. Top values:
 - LINEPROTO (28.63%)
 - ILPOWER (23.12%)
 - LINK (22.32%)
 - ACLLOG (16.12%)
 - ETHPORT (7.21%)
 - SFF8472 (0.6%)
 - FEX104 (0.5%)
 - FEX108 (0.5%)
 - PIM (0.5%)
 - DFC9 (0.2%)
- message** visualization: 999 documents (100%), 455 distinct values. Top values:
 - Interface Gi2/... (3.7%)
 - Interface Gi4/... (3%)
 - Interface Gi1/0... (2.8%)
 - Controller port... (2.4%)
 - Line protocol ... (1.6%)
 - Line protocol ... (1.6%)
 - Interface Fa3/... (1.5%)
 - Interface Gi1/0... (1.4%)
 - Interface Fa4/... (1.2%)
 - Interface Fa3/... (1.1%)
- #severity** visualization: 999 documents (100%), 12 distinct values.
- @timestamp** visualization: 999 documents (100%), 12 distinct values.

At the bottom, there are **Import** and **Cancel** buttons.

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.





The screenshot shows the Data Visualizer interface with the following details:

- Top Navigation:** Machine Learning, Job Management, Anomaly Explorer, Single Metric Viewer, Data Frames, **Data Visualizer** (underlined), Settings.
- Left Sidebar:** Icons for Home, Refresh, DataFrames, Metrics, Charts, Logstash, File, and Help.
- Main Area:**
 - Import data** (EXPERIMENTAL)
 - Simple** (selected) and **Advanced** tabs.
 - Index name:** sys6
 - Create index pattern:**
 - Import** button.

Click on Import, and you will see status:





The screenshot shows the Kibana Data Visualizer interface with the following details:

- Header:** Top navigation bar with tabs: Personal, Commercial, CNC, Cisco, Slack, VMware, Equinix, IP Journal, Work, Technical. Below it, specific sections like CNC Calendar, Test grok patterns, and Kibana are visible.
- Main Navigation:** Left sidebar with icons for Machine Learning, Job Management, Anomaly Explorer, Single Metric Viewer, Data Frames, **Data Visualizer** (which is selected and underlined), and Settings.
- Import Data Section:** A large central area titled "Import data" (EXPERIMENTAL). It includes a "Simple" tab (selected) and an "Advanced" tab. A dropdown menu for "Index name" is open, showing "sys6". A checked checkbox for "Create index pattern" is present. A prominent "Import" button is at the bottom.
- Status Summary:** Below the import section, a horizontal progress bar shows the status of five steps:
 - File processed (blue circle with a checkmark)
 - Index created (blue circle with a checkmark)
 - Ingest pipeline created (blue circle with a checkmark)
 - Uploading data (grey circle with the number 4)
 - Create index pattern (grey circle with the number 5)The "Uploading data" step is currently active, indicated by a blue underline.

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



**✓ Import complete**

Index	sys6
Index pattern	sys6
Ingest pipeline	sys6-pipeline
Documents ingested	36142
Failed documents	19

⌚ Some documents could not be imported

19 out of 36161 documents could not be imported. This could be due to lines not matching the Grok pattern.

> Failed documents



View index in Discover



Open in Data Visualizer



Index Management



Index Pattern Management

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):





The screenshot shows the Grok Constructor application running on a local host at `grokconstructor.appspot.com`. The interface includes a navigation bar with links to Personal, Commercial, CNC, Cisco, Slack, VMware, Equinix, IP Journal, Work, and Technical categories. Below the navigation is a secondary navigation bar with links to About, Incremental Construction, Matcher (which is highlighted), (New!) Pattern Translator, and Automatic Construction. The main content area has a heading 'Test grok patterns'. It contains a text input field for log lines, a 'Go!' button, and a text area for displaying matched patterns. A sidebar on the right provides a 'random example' button.

Test grok patterns

This tries to parse a set of given logfile lines with a given [grok regular expression](#) (based on [Oniguruma regular expressions](#)) and prints the matches for named patterns for each log line. You can also apply a [multiline filter](#) first.

Please enter some loglines for which you want to check a grok pattern, the grok expression that should match these, mark the pattern libraries you draw your patterns from and then press

Go!

You can also just try this out with a

random example

Some log lines you want to match. It's helps much to use several lines, and to choose lines that are as diverse as possible.

```
2018-05-18T00:00:00.105372-04:00 10.25.64.101 55305: 055358: May 17 23:59:59.100 EDT: %LINK-3-UPDOWN: Interface GigabitEthernet1/0/15, changed state to up
2018-05-18T00:00:00.106064-04:00 10.25.64.101 55306: 055359: May 18 00:00:00.100 EDT: %LINEPROTO-5-UPDOWN: Line protocol on Interface GigabitEthernet1/0/15, changed state to up
2018-05-18T00:00:00.106834-04:00 10.25.80.22 1058159: 1058063: May 17 23:59:59.098 EDT: %LINEPROTO-5-UPDOWN: Line protocol on Interface FastEthernet2/0/35, changed state to up
```

The (unquoted!) pattern that should match all logfile lines.(Please keep in mind that the whole log line / message is searched for this pattern; if you want this to match the whole line, enclose it in ^ \$ or \A \Z. This speeds up the search - especially if the pattern is not found.)

```
%{TIMESTAMP_ISO8601} %{IPV4} %{INT}?: %{INT}?: (%{INT})?: (%{INT})?: (%{INT})?: (%{INT})?: (%{INT})?: (%{INT})?: (%{INT})?: (%{INT})?: (%{INT})?: (%{WORD})-%{WORD}-%{WORD}: %{GREEDYDATA}
```

Please mark the libraries of [grok Patterns](#) from [logstash](#) v.2.4.0 which you want to use. You probably want to use grok-patterns if you use any of the others, since they rely on the basic patterns defined there.

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:





Grok Debugger Debugger Discover Patterns

```
2018-05-18T00:00:02.807814-04:00 10.25.217.11 411024: 612860: May 18 00:00:00.825 EDT: %ILPOWER-5-IEEE_DISCONNECT: Interface  
Fa3/0/1: PD removed (200dyer-bs-3)
```

```
%{TIMESTAMP_ISO8601:timestamp}.*?%{DATA:errcode}?:%{GREEDYDATA:errmsg}
```

Add custom patterns Keep Empty Captures Named Captures Only Singles

Autocomplete

Go

```
{
  "timestamp": [
    [
      "2018-05-18T00:00:02.807814-04:00"
    ]
  ],
  "YEAR": [
    [
      "2018"
    ]
  ],
  "MONTHNUM": [
    [
      "05"
    ]
  ],
  "MONTHDAY": [
    [
      "18"
    ]
  ],
  "HOUR": [
    [
      "00",
      "04"
    ]
  ],
}
```

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 Engine 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.

