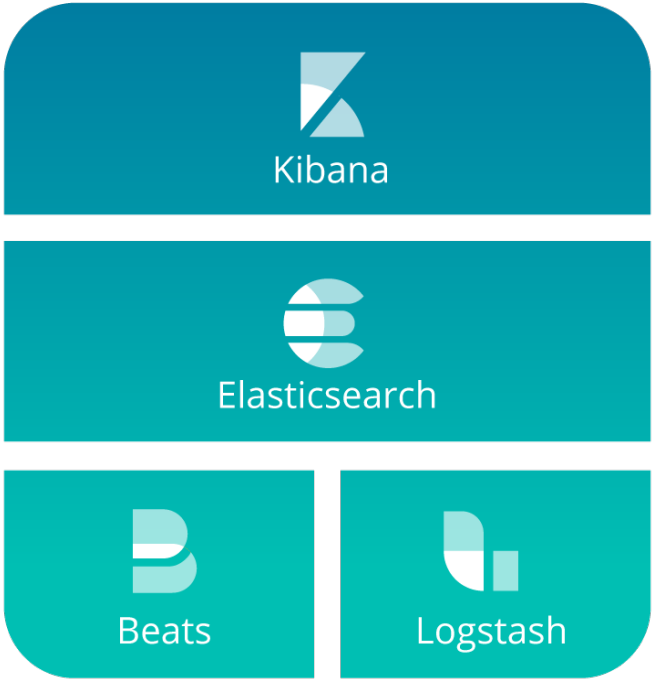
From Raw Logs to Real Insights

Getting started with log analytics using the Elastic Stack

Rob Cowart
Solutions Architect

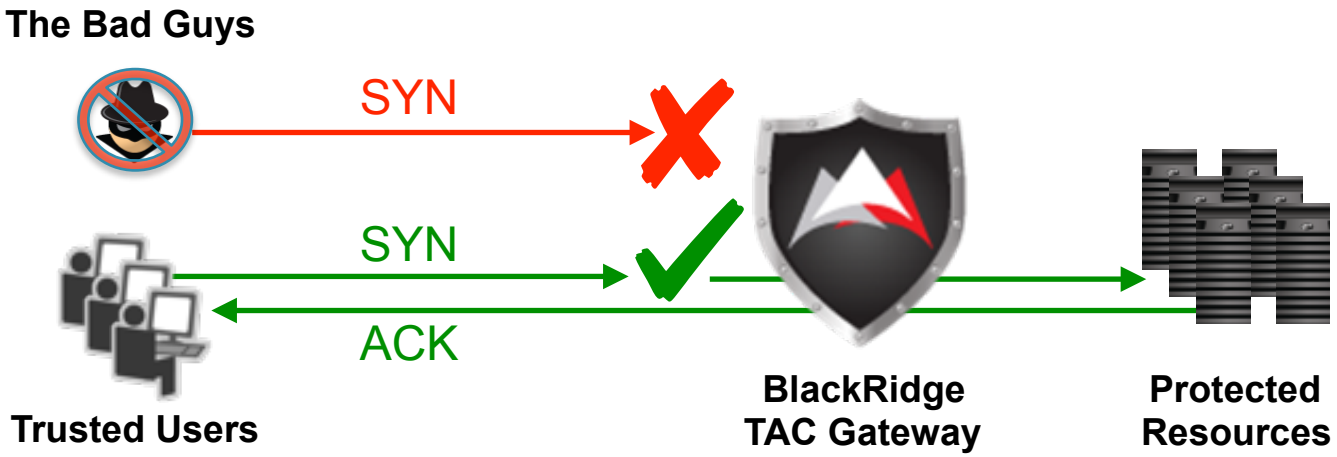
Elastic Stack

100% open source
No enterprise edition



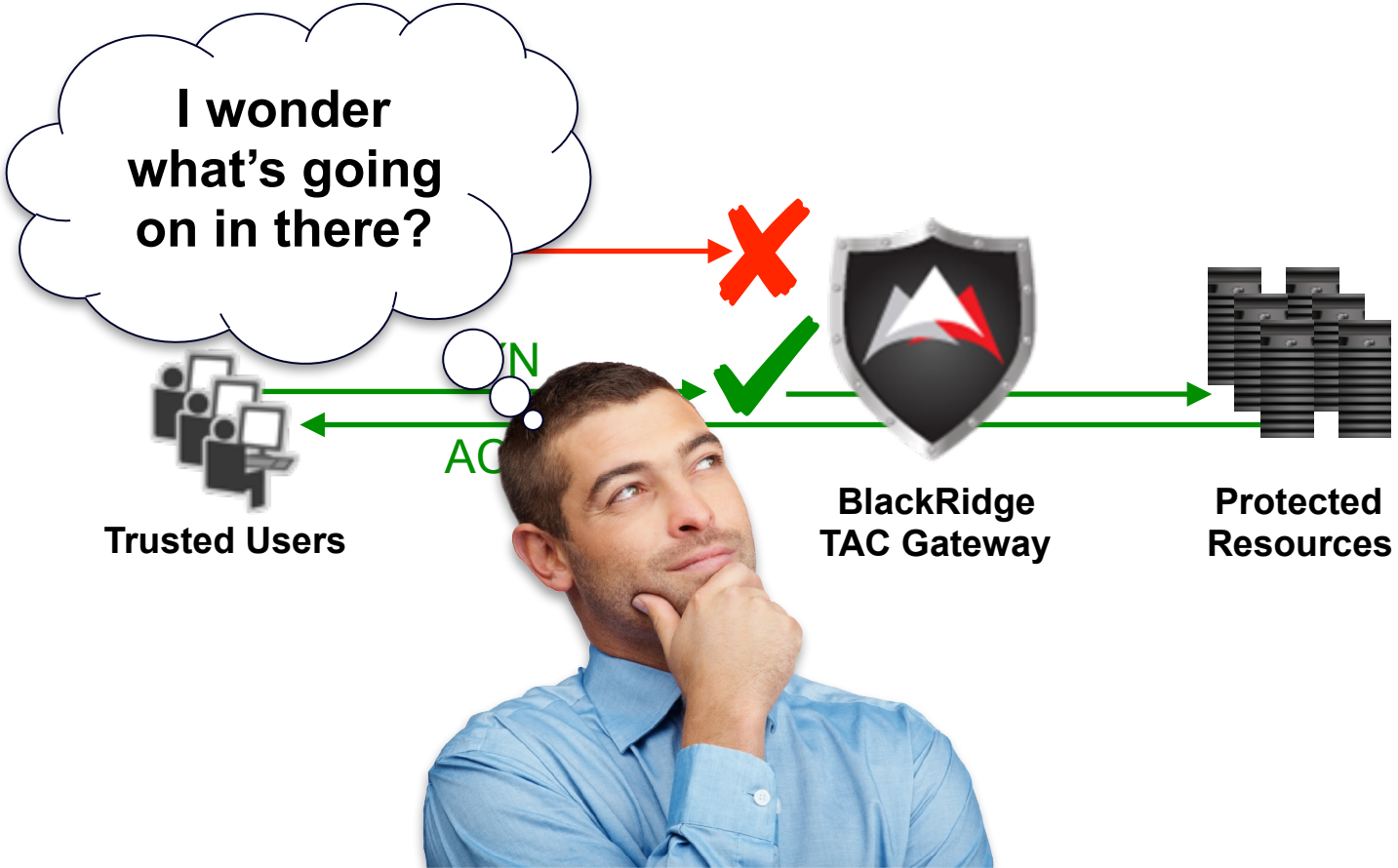
elastic

Our sample data… from the BlackRidge TAC Gateway



elastic

Our sample data… from the BlackRidge TAC Gateway



elastic

A look at BlackRidge Message Format (BMF)

<5>Oct 18 12:57:30 BRDC-2 kernel: [BlackRidge|Gateway|3.0.0.4619] class="Attribution" category="Unknown Identity" ctx="bump0" src="125.33.12.234" srcPort="25654" dest="5.149.112.53" destPort="23" identity="honeypot2Id" gwAction="DISCARD" gwMode="Monitor"

What's included in BlackRidge Message Format (BMF)?

Syslog header:

<5>Oct 18 12:57:30 BRDC-2 kernel:

BMF Header:

[BlackRidge|Gateway|3.0.0.4619]

BMF Payload:

class="Attribution" category="Unknown Identity" ctx="bump0" src="125.33.12.234" srcPort="25654" dest="5.149.112.53" destPort="23" identity="honeypot2Id" gwAction="DISCARD" gwMode="Monitor"

Ingesting data into the Elastic Stack

pipeline

parse, format and enrich the data

index template

specify how the data should be indexed

index pattern & dashboards

view and analyze the data

Building the Pipeline

The structure of a Logstash pipeline

```
input {

}

filter {

}

output {

}
```

From where we get the data.

How we parse, format and enrich the data.

Where we send the data.

Listen for syslog but read from a file for development

```
input {
  udp {
    type => "syslog"
    port => "${ESLOG_SYSLOG_PORT:514}"
  }

  tcp {
    type => "syslog"
    port => "${ESLOG_SYSLOG_PORT:514}"
  }

  file {
    path => "${ESLOG_BASE}/logs/dev.syslog"
    sincedb_path => "/dev/null"
    start_position => "beginning"
    ignore_older => 0
  }
}
```

Environment variables can make pipelines more portable.

We will use sample data from a file as we develop the pipeline.

These settings tell logstash to reread the file each time it is run.

This is for development, NOT production.

Environment Variables Used in this Tutorial

```
#=================== ES Log Tutorial environment variables ====================
export ESLOG_BASE=/PATH/TO/eslog_tutorial
export ESLOG_DICTIONARY_PATH=$ESLOG_BASE/logstash/dictionaries
export ESLOG_GROK_PATTERNS_DIR=$ESLOG_BASE/logstash/patterns
export ESLOG_GEOIP_DBS_DIR=$ESLOG_BASE/logstash/geoipdbs
export ESLOG_TEMPLATE_PATH=$ESLOG_BASE/logstash/templates
export ESLOG_LOGS=$ESLOG_BASE/logs

export ESLOG_ELASTICSEARCH_HOSTS=127.0.0.1:9200
export ESLOG_ELASTICSEARCH_USER=elastic
export ESLOG_ELASTICSEARCH_PASSWORD=changeme

export ESLOG_SYSLOG_PORT=514
export ESLOG_SYSLOG_TZ=UTC
#================ end of ES Log Tutorial environment variables ================
```

Edit as needed for your environment.

Output to stdout

```
output {
  stdout {
    codec => rubydebug {
      metadata => true
    }
  }
}
```

For now we only care about seeing the results of our pipeline configuration, so we will send the data to stdout.

The rubydebug codec formats the output for better readability

By default @metadata fields are not output, but we can specify that we want to see them.

Run Logstash (01_blackridge.logstash.conf)

```
LOGSTASH_HOME/bin/logstash --path.config $ESLOG_BASE/logstash/01_blackridge.logstash.conf
```

RESULT:

```
{
  "path" => "/Users/rob/src/eslog_tutorial/logs/dev.syslog",
  "@timestamp" => 2017-10-23T06:02:29.528Z,
  "@metadata" => {
    "path" => "/Users/rob/src/eslog_tutorial/logs/dev.syslog",
    "host" => "ws5.local"
  },
  "@version" => "1",
  "host" => "ws5.local",
  “message" => "<5>Oct 18 12:57:30 BRDC-2 kernel: [BlackRidge|Gateway|3.0.0.4619] class=\"Attribution\"
category=\"Unknown Identity\" ctx=\"bump0\" src=\"125.33.12.234\" srcPort=\"25654\" dest=\"5.149.112.53\"
destPort=\"23\" identity=\"honeypot2Id\" gwAction=\"DISCARD\" gwMode=\"Monitor\""
}
```

It would be better if this was the the timestamp from the received message.

This is the host that forwarded the message, not necessarily from where it originated.

message is the raw data that was received.

Does this look like a syslog message?

```
filter {
  grok {
    match => {
      "message" => "(?:<%{INT:syslog_pri}>\s*)?(?:%{SYSLOGTIMESTAMP:[@metadata][syslog_timestamp]}|%
{TIMESTAMP_ISO8601:[@metadata][syslog_timestamp]})\s+(?:%{SYSLOGFACILITY}\s+)?%{IPORHOST:logging_host}\s+%
{PROG:logging_process}(?:\[%{INT:logging_pid}\])?:\s+%{GREEDYDATA:logged_message}"
    }
  }
  if "_grokparsefailure" in [tags] {
    drop { }
  }
}
```

Let's see if this looks like a syslog message

This will be the BMF formatted message

This is from where the message originated

We will drop messages that don't look like properly formatted syslog. You may want to handle them differently depending on your needs.

TO BE CONTINUED

Run Logstash (02_blackridge.logstash.conf)

```
LOGSTASH_HOME/bin/logstash --path.config $ESLOG_BASE/logstash/02_blackridge.logstash.conf
```

RESULT: *only new fields are shown*

```
"logging_host" => "BRDC-2",
"logging_process" => "kernel",
"@metadata" => {
  "syslog_timestamp" => "Oct 18 12:57:30"
},
"logged_message" => "[BlackRidge|Gateway|3.0.0.4619] class=\"Attribution\" category=\"Unknown Identity\"
ctx=\"bump0\" src=\"125.33.12.234\" srcPort=\"25654\" dest=\"5.149.112.53\" destPort=\"23\"
identity=\"honeypot2Id\" gwAction=\"DISCARD\" gwMode=\"Monitor\"",
“syslog_pri" => "5"
```

We can use this to set *@timestamp*

syslog_pri can be further decoded

The BMF formatted message, which we will process further.

elastic

Decode *syslog_pri*

```
else {
  syslog_pri { }
}
```

It looks like we got a syslog message so let's decode *syslog_pri*.

Since we stored the syslog priority value in a field named *syslog_pri*, we don't have to specify any options.

TO BE CONTINUED

Run Logstash (03_blackridge.logstash.conf)

```
LOGSTASH_HOME/bin/logstash --path.config $ESLOG_BASE/logstash/03_blackridge.logstash.conf
```

RESULT: *only new fields are shown*

```
"syslog_facility_code" => 0,
"syslog_facility" => "kernel",
"syslog_severity_code" => 5,
"syslog_severity" => "notice",
```

You may want to use the color formatter within the Kibana index pattern definition to display different severity values in different colors.

Set @timestamp to the time from the raw message

```
date {
  locale => "en"
  match => [ "[@metadata][syslog_timestamp]", "MMM  d HH:mm:ss", "MMM dd HH:mm:ss", "MMM dd YYYY HH:mm:ss",
"MMM  d YYYY HH:mm:ss", "ISO8601" ]
  timezone => "${ESLOG_SYSLOG_TZ:UTC}"
}
```

TO BE CONTINUED

Ideally we would always get a timestamp in UTC, but just in case we need to change it, let's make it configurable via an environment variable.

Syslog timestamps can come in a lot of different formats. Specify the common ones here to make the pipeline more adaptable.

Run Logstash (04_blackridge.logstash.conf)

```
LOGSTASH_HOME/bin/logstash --path.config $ESLOG_BASE/logstash/04_blackridge.logstash.conf
```

RESULT: *only new fields are shown*

```
"@timestamp" => 2017-10-18T12:57:30.000Z
```

@timestamp is now set to the timestamp from the raw syslog message.

Everything up to this point was handling of the syslog-specific aspects of the message. You may want to reuse this as the foundation for all of your syslog sources.

Does this look like a BlackRidge BMF message?

```
grok {
  patterns_dir => [ "${ESLOG_GROK_PATTERNS_DIR:/etc/logstash/patterns}" ]
  match => { "logged_message" => "%{BMF_MSG}" }
  add_tag => [ "blackridge" ]
}
if "_grokparsefailure" in [tags] {
  drop { }
}
```

This time we are using an external grok patterns file instead of an inline pattern.

BMF_MSG is defined in an external patterns file.

```
BMF_MSG ^\[[BL][lE][aE][cF]%{GREEDYDATA}\]\s+%{GREEDYDATA:[@metadata][bmf_payload]}$
```

TO BE CONTINUED

Run Logstash (05_blackridge.logstash.conf)

```
LOGSTASH_HOME/bin/logstash --path.config $ESLOG_BASE/logstash/05_blackridge.logstash.conf
```

RESULT: *only new fields are shown*

```
"@metadata" => {
  "bmf_payload" => "class=\"Attribution\" category=\"Unknown Identity\" ctx=\"bump0\" src=\"125.33.12.234\"
srcPort=\"25654\" dest=\"5.149.112.53\" destPort=\"23\" identity=\"honeypot2Id\" gwAction=\"DISCARD\"
gwMode=\”Monitor\""
},
"tags" => [
  [0] "blackridge"
],
```

The *@metadata.bmf_payload* field now contains the key-value portion of the BMF message.

Tags are useful for many purposes, including controlling the how the message is processed.

Transform the key-value pairs into fields

```
else {
  kv {
    source => "[@metadata][bmf_payload]"
    trim_key => "\""
    trim_value => "\""
  }
}
```

The *kv* filter adds a field for each key-value pair.

We can use the *trim_key* and *trim_value* options to remove the quotes from the message

TO BE CONTINUED

Run Logstash (06_blackridge.logstash.conf)

```
LOGSTASH_HOME/bin/logstash --path.config $ESLOG_BASE/logstash/06_blackridge.logstash.conf
```

RESULT: *only new fields are shown*

```
"src" => "125.33.12.234",
"srcPort" => “25654”,
"dest" => "5.149.112.53",
"destPort" => “23",
"class" => "Attribution",
"category" => "Unknown Identity",
"ctx" => "bump0",
"gwAction" => "DISCARD",
"gwMode" => "Monitor",
"identity" => "honeypot2Id",
```

Wouldn’t it be interesting to know where this IP is located?

Do users know which service is being accessed? We can make this more user-friendly.

The *gwAction* field is another good candidate for the color formatter in Kibana.

Was the access attempt internal or external?

```
if [src] {
  cidr {
    address => [ "%{src}" ]
    network => [ "10.0.0.0/8", "172.16.0.0/12", "192.168.0.0/16", "fc00::/7", "127.0.0.0/8", "::
1/128","169.254.0.0/16", "fe80::/10","224.0.0.0/4", “ff00::/8", "255.255.255.255/32" ]
    add_field => { "src_locality" => "private" }
  }
}
```

TO BE CONTINUED

The *cidr* filter determines if an IP address is within a specified subnet. Let’s check all private IP blocks.

Run Logstash (07_blackridge.logstash.conf)

```
LOGSTASH_HOME/bin/logstash --path.config $ESLOG_BASE/logstash/07_blackridge.logstash.conf
```

RESULT: *only new fields are shown*

```
"src_locality" => "private"
```

If the *src* is in fact a private IP then you would see the above output. However in the sample event we are working with the *src* is a public IP address.

From where did the access attempt come?

```
if ![src_locality] {
  mutate {
    add_field => { "src_locality" => "public" }
  }
  geoip {
    source => "src"
    database => "${ESLOG_GEOIP_DBS_DIR:/etc/logstash/geoipdbs}/GeoLite2-City.mmdb"
    target => "geoip"
  }
  geoip {
    source => "src"
    database => "${ESLOG_GEOIP_DBS_DIR:/etc/logstash/geoipdbs}/GeoLite2-ASN.mmdb"
    target => "geoip"
  }
}
```

Since the *src_locality* isn't private it must be public.

Lookup the IP in the City DB for its geo-location.

Lookup the IP in the ASN DB to determine the public network space (the autonomous system) from which it originated.

TO BE CONTINUED

The *geoip* filter includes the City and ASN DBs, but if you want to update them more regularly or use the commercial City and ISP DBs it is useful to maintain them in their own directory.

Run Logstash (08_blackridge.logstash.conf)

```
LOGSTASH_HOME/bin/logstash --path.config $ESLOG_BASE/logstash/08_blackridge.logstash.conf
```

RESULT: *only new fields are shown*

```
"geoip" => {
  “ip" => "125.33.12.234",
  "longitude" => 116.3883,
  "latitude" => 39.9289,
  "continent_code" => "AS",
  "city_name" => "Beijing",
  "country_name" => "China",
  "country_code2" => "CN",
  "country_code3" => "CN",
  "region_code" => "11",
  "region_name" => "Beijing",
  "timezone" => “Asia/Shanghai",
  "location" => {
    "lon" => 116.3883,
    "lat" => 39.9289
  },
  "as_org" => "China Unicom Beijing Province Network",
  "asn" => 4808
}
```

The geoip.country_code2 field will allow us to visualize the data on Kibana’s vector map visualization.

The *geoip.location* object will allow us to visualize the data on a tilemap.

Let's make the *destPort* more user-friendly!

```
if [src] {
  ...
}

if [destPort] {
  translate {
    dictionary_path => "${ESLOG_DICTIONARY_PATH:/etc/logstash/dictionaries}/iana_service_names_tcp.yml"
    field => "destPort"
    destination => "service"
    fallback => "__UNKNOWN"
  }
  if [service] == "__UNKNOWN" {
    mutate {
      replace => { "service" => "%{[destPort]}" }
    }
  } else {
    mutate {
      replace => { "service" => "%{[service]} (%{[destPort]})" }
    }
  }
}
```

If the *destPort* exists let's create a more user-friendly *service* field.

The translate filter can lookup values in an external dictionary file. Here we use one made from IANA's registry of common services.

If we don't find a service name in the dictionary then let's set the *service* to the *destPort*.

If we do find a service name in the dictionary then let's construct the *service* to from the service name and the *destPort*.

Run Logstash (09_blackridge.logstash.conf)

```
LOGSTASH_HOME/bin/logstash --path.config $ESLOG_BASE/logstash/09_blackridge.logstash.conf
```

RESULT: *only new fields are shown*

```
"service" => "telnet (23)"
```

The resulting *service* field provides a much more user-friendly representation than the *destPort*.

Time to send our data to Elasticsearch

```
output {
  #stdout {
  #  codec => rubydebug {
  #    metadata => true
  #  }
  #}

  if "blackridge" in [tags] {
    elasticsearch {
      hosts => [ "${ESLOG_ELASTICSEARCH_HOSTS:127.0.0.1:9200}" ]
      user => "${ESLOG_ELASTICSEARCH_USER}"
      password => "${ESLOG_ELASTICSEARCH_PASSWORD}"
      index => "blackridge-%{+YYYY.MM.dd}"
    }
  }
}
```

Comment out the *stdout* output.

Only send to Elasticsearch if it was identified as a BlackRidge BMF message.

Specify the Elasticsearch server to connect to, including any needed credentials.

Specify the index name. Here we are using daily indexes.

Run Logstash (10_blackridge.logstash.conf)

```
LOGSTASH_HOME/bin/logstash --path.config $ESLOG_BASE/logstash/10_blackridge.logstash.conf
```

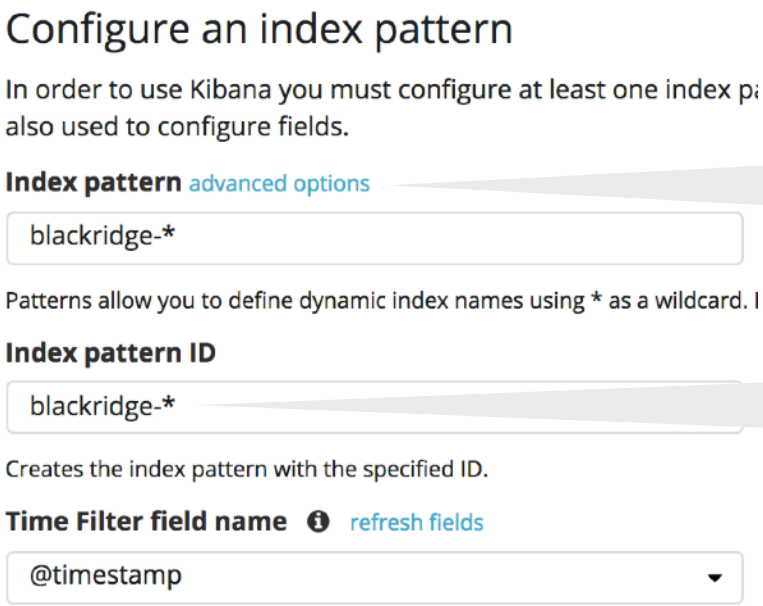
RESULT:

You should not see an output to stdout since we commented that out. However you should have data in Elasticsearch (one document anyway).

So let’s move forward and take a first look at the document we have created in Kibana, and begin our discussion of Index Templates.

Designing Index Templates

Create a Kibana Index Pattern

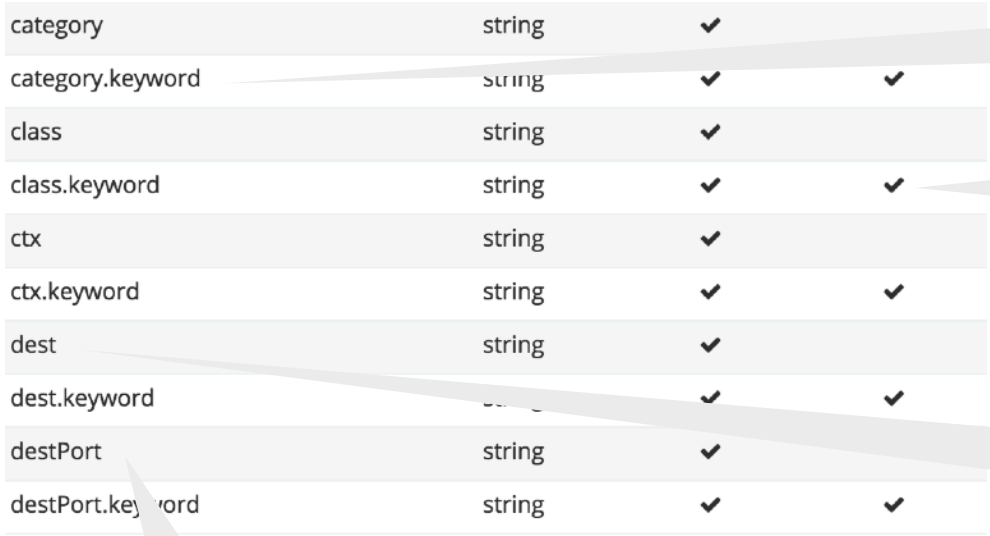


Click *advance options* in order to be able to specify an Index pattern ID (necessary as of 5.6)

For convience make the Index pattern ID the same as the Index pattern title.

elastic

Create a Kibana Index Pattern



category	string	✔	
category.keyword	string	✔	✔
class	string	✔	
class.keyword	string	✔	✔
ctx	string	✔	
ctx.keyword	string	✔	✔
dest	string	✔	
dest.keyword		✔	✔
destPort	string	✔	
destPort.key ord	string	✔	✔

What is the deal with all of these *keyword* fields?

… and why are they the only ones that are aggregatable??

Shouldn't the *src* and *dest* be IP addresses?

Should *srcPort* and *destPort* be strings?

elastic

Index Templates

- It is often more convenient to rely on index autocreation than to have to create indices manually.
- Usually we want some control over the settings and mappings of the new index.
  - # of shards
  - # of replicas
  - The data for each field
  - What to do with unspecified fields (dynamic mappings)
- **Index templates** can be used to control which settings should be applied to newly created indices.

The structure of an Index Template

```
{
  "order": 0,
  "version": 17102801,
  "template": "blackridge-*",
  "settings": {},
  "mappings": {
    "_default_": {
      "dynamic_templates": [],
      "properties": {}
    }
  },
  "aliases": {}
}
```

Multiple templates may apply to a single index. Here we can specify the order that they are evaluated.

The version of the template. I like to use YYMMdd##

The index name pattern to which this template will be applied.

Index settings go here.

Configuration for dynamically created fields.

Alternative name by which the index can be accessed.

Configuration for static field mappings.

Index Settings

```
"settings": {
  "index": {
    "number_of_shards": 2,
    "number_of_replicas": 1,
    "refresh_interval": "5s",
    "codec": "best_compression"
  }
},
```

The number of shards that the index should have.

The number of replicas of each shard.

Set a refresh interval of 5s. Newly indexed documents will be searchable after 5 secs.

Use best compression save disk space.

Properties (static field mappings)

```
"properties": {
  "@timestamp": { "type": “date" },
  "@version": { "type": “keyword" },
  "geoip": {
    “properties": {
      "asn": { “type": “keyword" },
      "ip": { "type": “ip" },
      "latitude": { "type": “float" },
      "location": { "type": “geo_point" },
      ...
    }
  },
  ...
  "src": { "type": “ip" },
  "srcPort": { "type": “integer" }
  ...
}
```

date type is for timestamps.

Multi-level objects, such as *geoip,* are properties that contain properties.

keywords are non-analyzed strings. Can be used for aggregations.

float is for floating point values. (alt. *double*)

geo_point contain long/lat coordinates for locating data on a tilemap

ip is for both IPv4 and IPv6 addresses.

integer is 32-bit signed whole numbers. (alt. *byte, short, long*)

Dynamic Templates

```
"dynamic_templates": [
  {
    "string_fields": {
      "match_mapping_type": "string",
      "match": "*",
      "mapping": { "type": "keyword" }
    }
  }
],
```

Control how Elasticsearch uses dynamic mapping to determine the type of a previously unseen field.

If the field is a string…

… and it matches this pattern…

… then it is of type *keyword*.

Loading the Index Template from Logstash

```
elasticsearch {
  hosts => [ "${ESLOG_ELASTICSEARCH_HOSTS:127.0.0.1:9200}" ]
  user => "${ESLOG_ELASTICSEARCH_USER}"
  password => "${ESLOG_ELASTICSEARCH_PASSWORD}"
  index => "blackridge-%{+YYYY.MM.dd}"
  template => "${ESLOG_TEMPLATE_PATH:/etc/logstash/templates}/blackridge.template.json"
  template_name => "blackridge"
  template_overwrite => "true"
}
```

Load the template stored in this file.

Name the template as specified here.

If the template already exists, overwrite it with one being loaded.

Cleanup the previous attempt

```
curl -XDELETE http://127.0.0.1:9200/blackridge-2017.10.18 -u elastic
Enter host password for user ‘elastic': ********
{“acknowledged":true}
```

Specify a user if security is enabled.

Deletes the previously created index.

Run Logstash (11_blackridge.logstash.conf)

```
LOGSTASH_HOME/bin/logstash --path.config $ESLOG_BASE/logstash/11_blackridge.logstash.conf
```

RESULT:

You should not see an output to stdout since we commented that out. However you should have data in Elasticsearch (one document anyway).

Delete and Recreate the Kibana index pattern

category	string	✔	✔
class	string	✔	✔
ctx	string	✔	✔
dest	ip	✔	✔
destPort	number	✔	✔
geoip.as_org	string	✔	✔
geoip.asn	string	✔	✔
geoip.city_name	string	✔	✔

We now have clean field names without duplicate analyzed and non-analyzed fields.

As keywords these fields are aggregatable.

src and *dest* are now IP addresses.

srcPort and *destPort* are now numbers?

Load messages from data.syslog

```
input {
  udp {
    type => "syslog"
    port => "${ESLOG_SYSLOG_PORT:514}"
  }

  tcp {
    type => "syslog"
    port => "${ESLOG_SYSLOG_PORT:514}"
  }

  file {
    path => “${ESLOG_BASE}/logs/data.syslog"
    sincedb_path => "/dev/null"
    start_position => "beginning"
    ignore_older => 0
  }
}
```

Edit the *file* input to read from *data.syslog*

Run Logstash (12_blackridge.logstash.conf)

```
LOGSTASH_HOME/bin/logstash --path.config $ESLOG_BASE/logstash/12_blackridge.logstash.conf
```

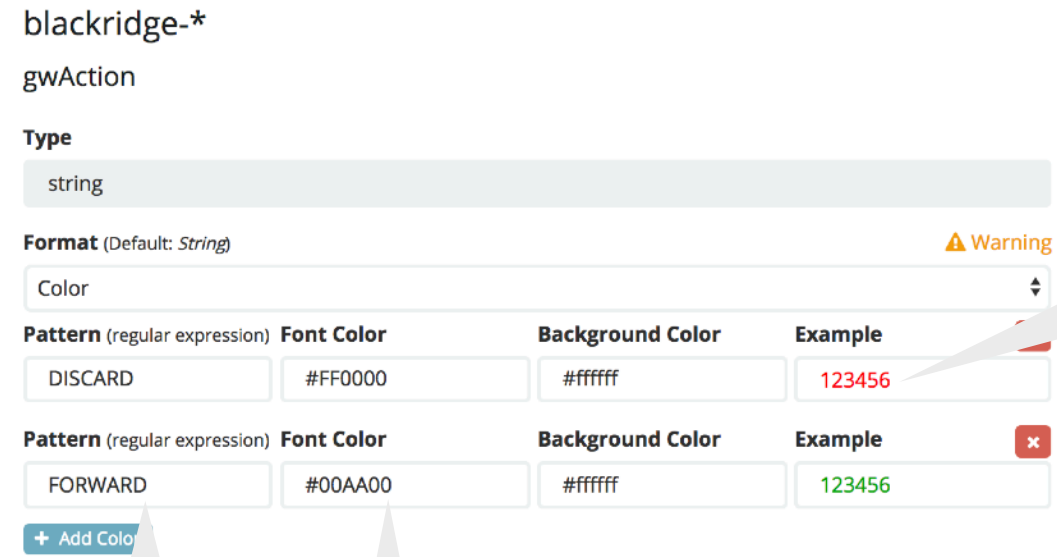
RESULT:

You should not see an output to stdout since we commented that out. However you should have data in Elasticsearch (one document anyway).

Follow in the Kibana *discover* app as the data is loaded. When complete you will have 7 days of data from 18-24 Oct.

Fine Tuning the Index Pattern

Using the Color Field Formatter



The result will look like this.

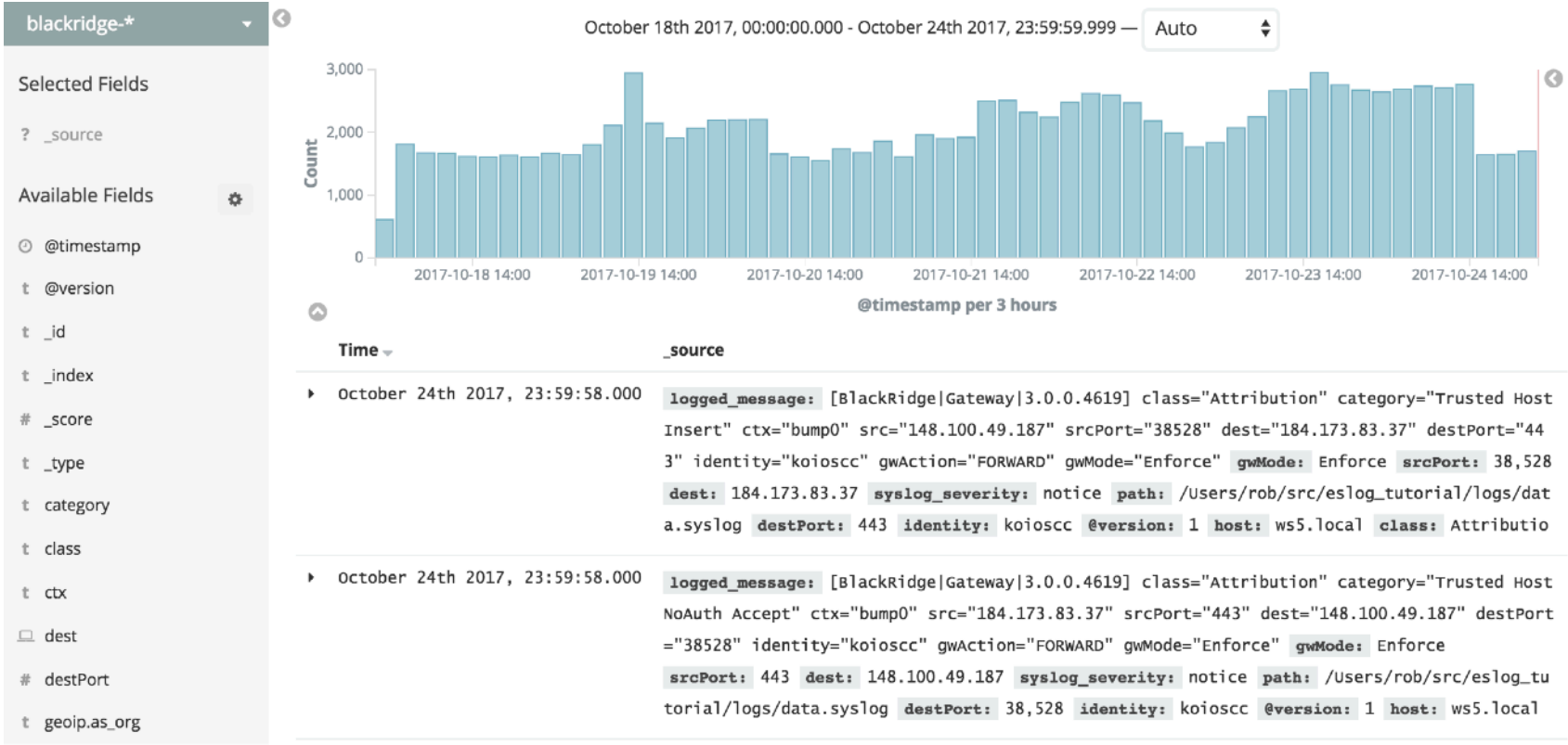
For each possible value of the field…

… you can specify a color to be used whenever the value is displayed.

elastic

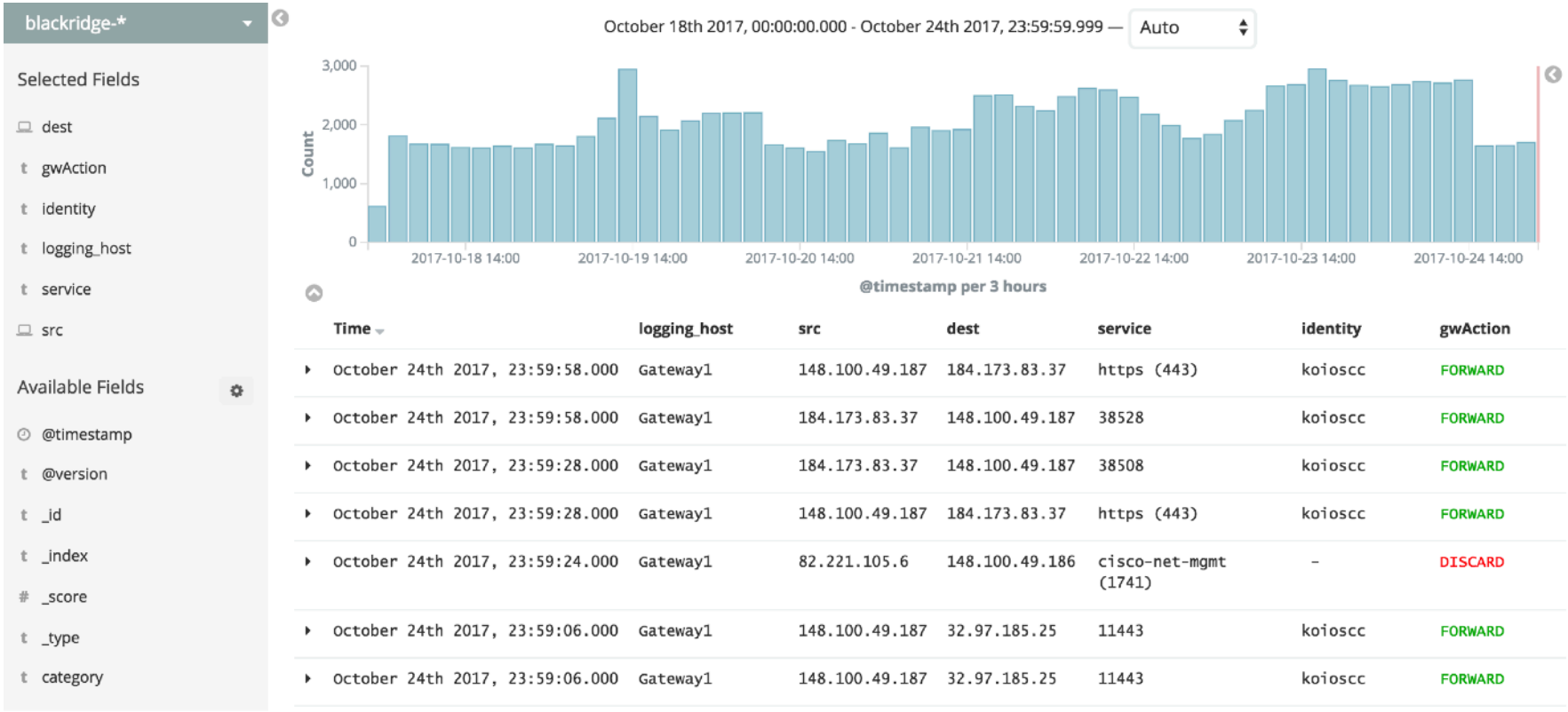
Building Kibana Dashboards

Using discover



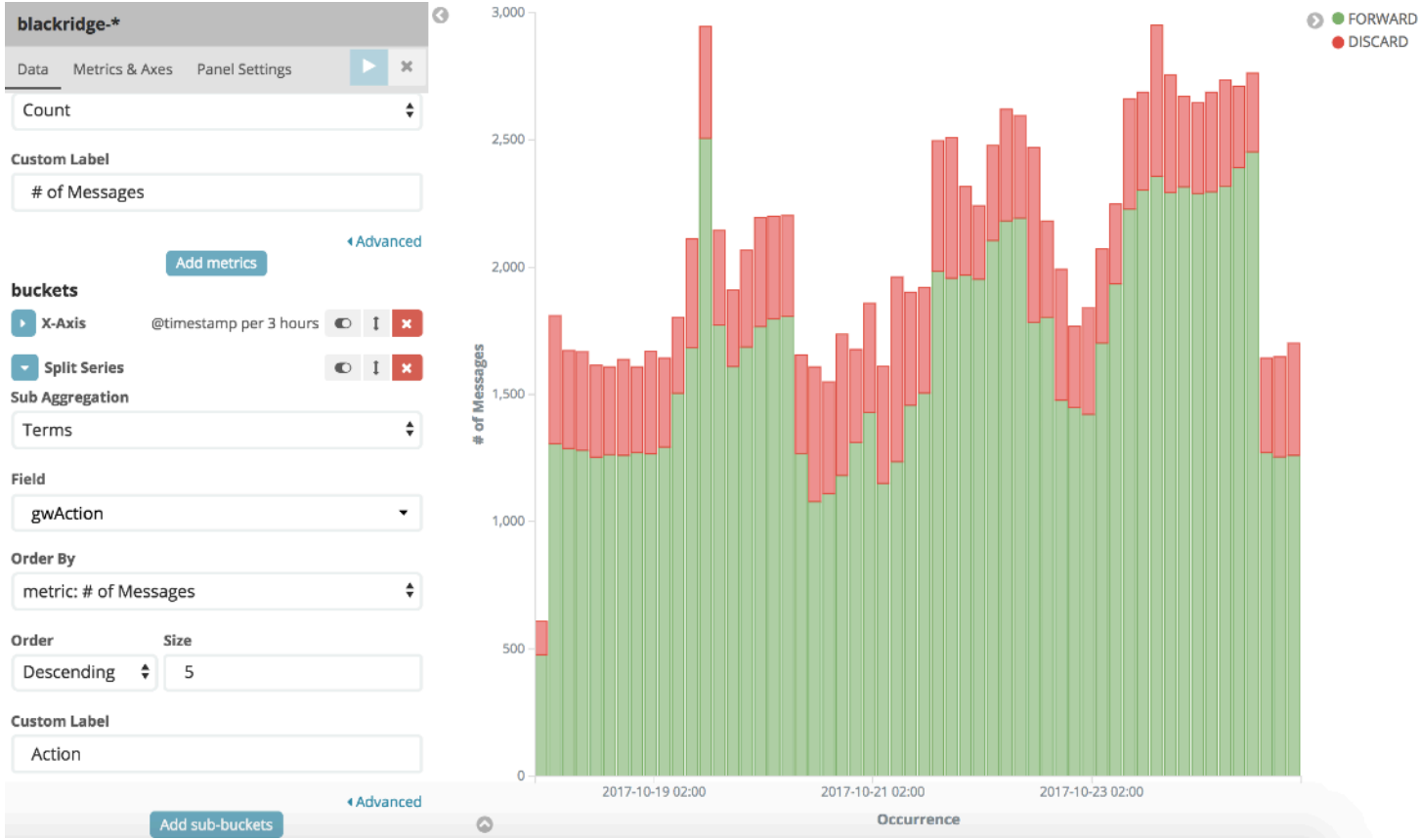
elastic

Creating a Saved Search



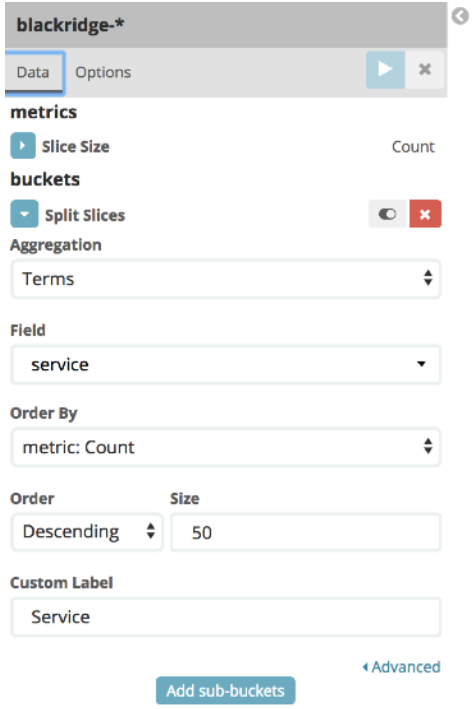
elastic

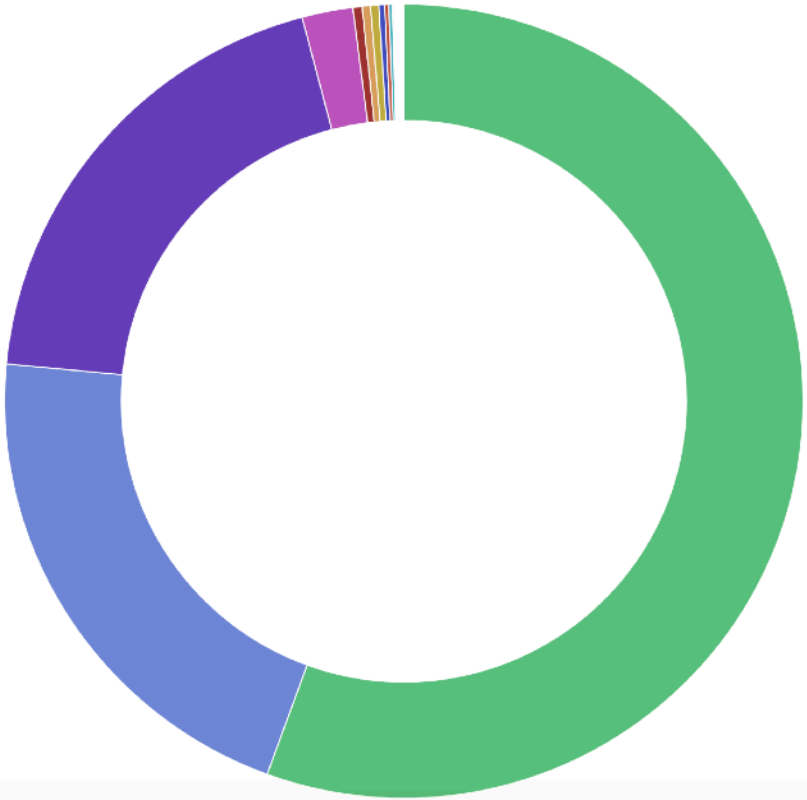
Creating a Vertical Bar Chart

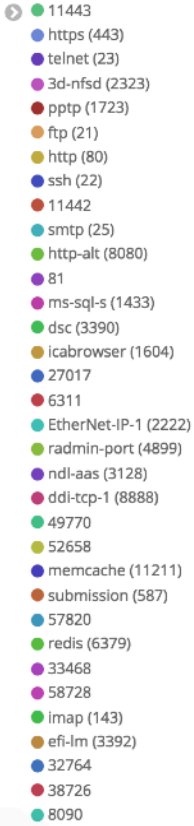


elastic

Creating a Pie or Donut Chart







elastic

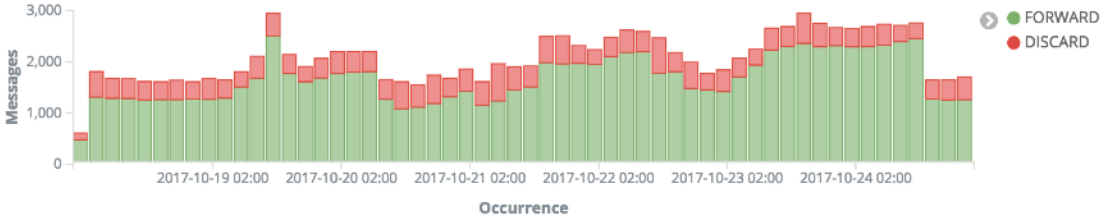
Combining Visualizations to Create Dashboards

BlackRidge: Message Count

Messages

116,698

BlackRidge: Messages by Action



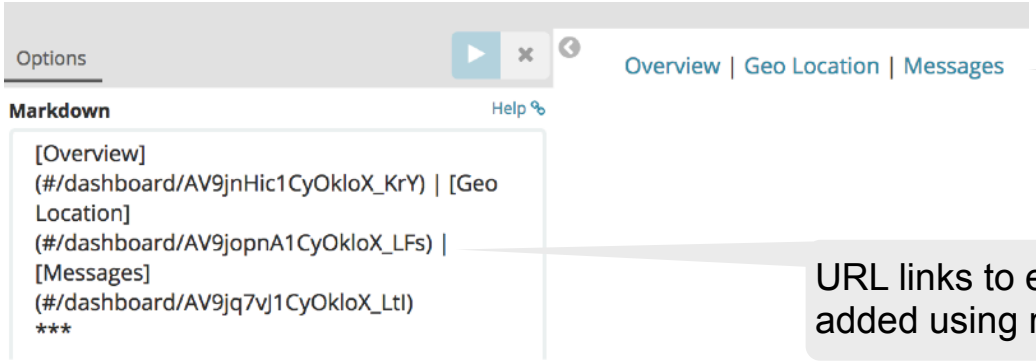
BlackRidge: Messages

1–50 of 116,698

Time	logging_host	src	dest	service	identity	gwAction
October 24th 2017, 23:59:58.000	Gateway1	148.100.49.187	184.173.83.37	https (443)	koioscc	FORWARD
October 24th 2017, 23:59:58.000	Gateway1	184.173.83.37	148.100.49.187	38528	koioscc	FORWARD
October 24th 2017, 23:59:28.000	Gateway1	148.100.49.187	184.173.83.37	https (443)	koioscc	FORWARD
October 24th 2017, 23:59:28.000	Gateway1	184.173.83.37	148.100.49.187	38508	koioscc	FORWARD
October 24th 2017, 23:59:24.000	Gateway1	82.221.105.6	148.100.49.186	cisco-net-mgmt (1741)	-	DISCARD
October 24th 2017, 23:59:06.000	Gateway1	148.100.49.187	32.97.185.25	11443	koioscc	FORWARD
October 24th 2017, 23:59:06.000	Gateway1	148.100.49.187	32.97.185.25	11443	koioscc	FORWARD
October 24th 2017, 23:59:06.000	Gateway1	148.100.49.187	32.97.185.25	11443	koioscc	FORWARD
October 24th 2017, 23:58:58.000	Gateway1	148.100.49.187	184.173.83.37	https (443)	koioscc	FORWARD

elastic

Using Markdown for Navigation between Dashboards



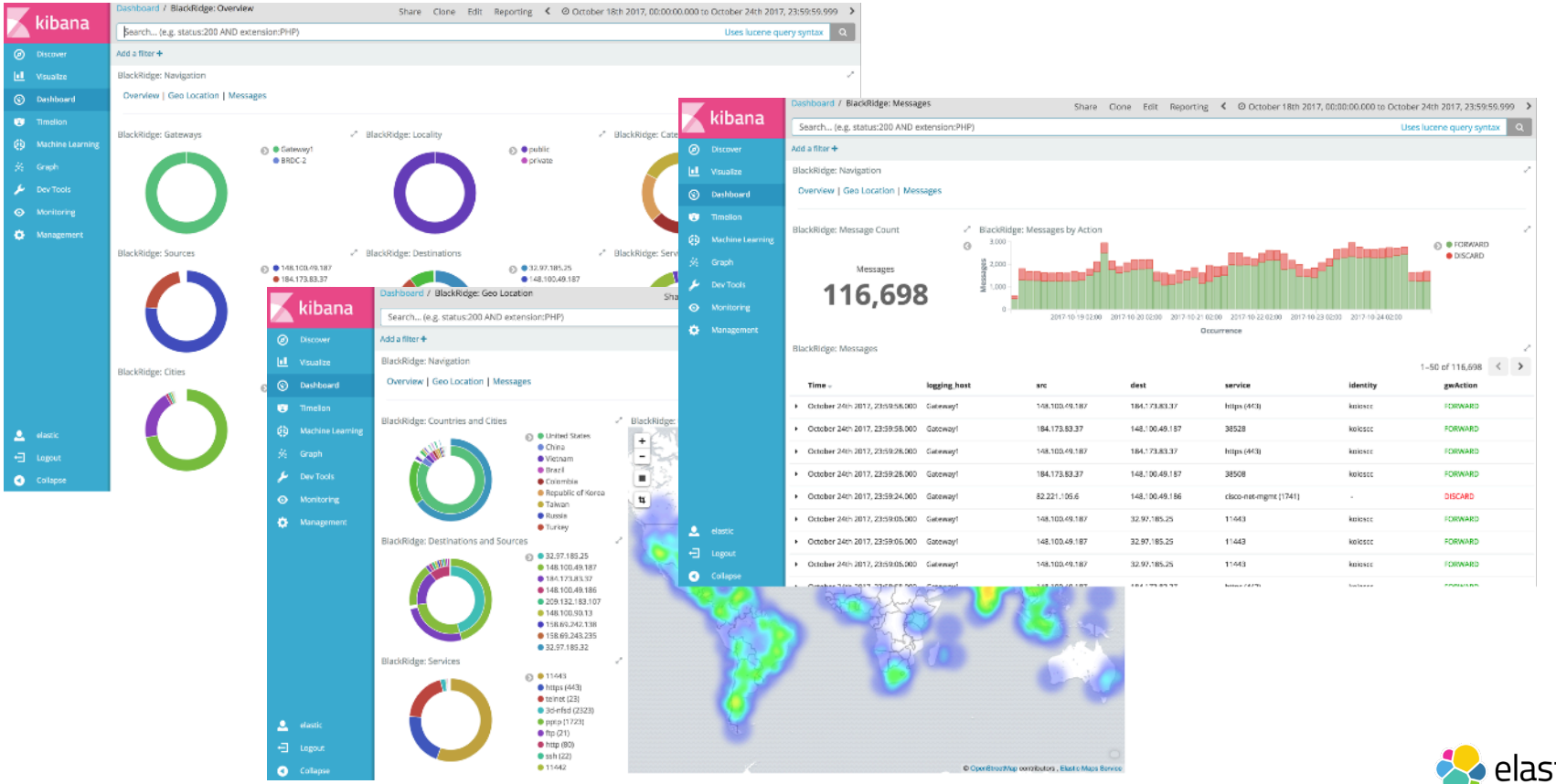
The resulting visualization can be added to dashboard to provide navigation capabilities.

URL links to each dashboard are added using markdown syntax.

Setting the Kibana advanced option *filters:pinnedByDefault* to *true* will allow filters to easily be maintained when switching dashboards.

elastic

From Raw Logs to Real Insights… with the Elastic Stack



elastic

Extra Credit…

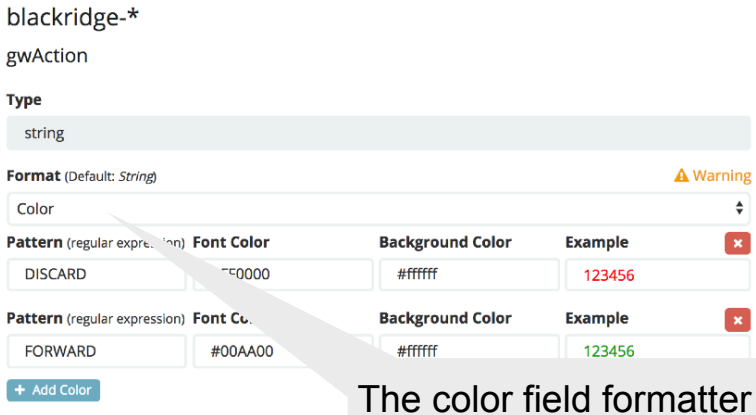
Add exception handling to the Logstash Pipeline

```
if "_grokparsefailure" in [tags] {
  drop { }
}
```

Twice in the Logstash pipeline we simply dropped messages that didn't match the expected format for which we tested.

CHALLENGE: Add handling for these exceptions such as sending these messages to "unrecognized messages" index, or write them to an a file so that they may be reviewed and better supported in future pipelines.

Using the Color Field Formatter



The color field formatter helped us to more easily identify conditions of concern.

CHALLENGE: Use the color field formatter to add similarly enhance the *syslog_severity* field.

elastic

Cleanup the Indexed Data

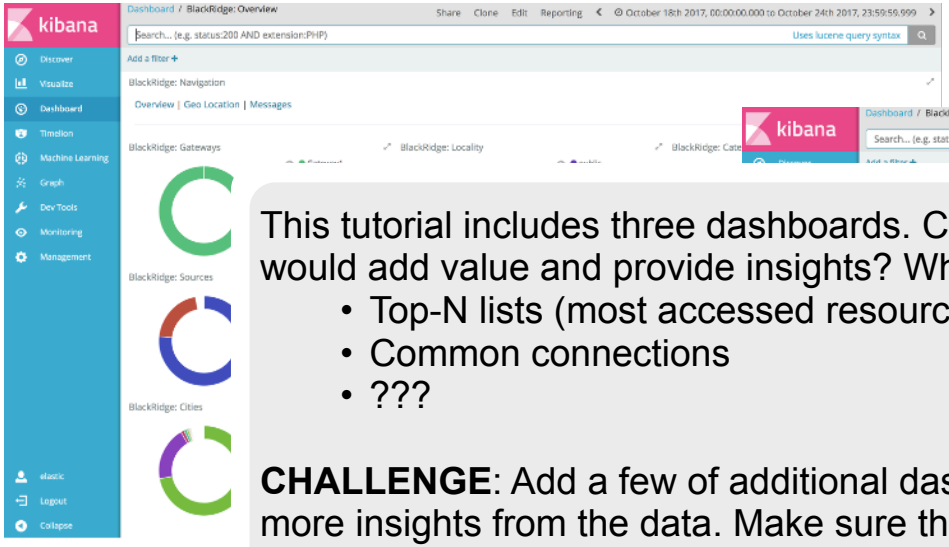
```
"geoip" => {
  “ip" => "125.33.12.234",
  "longitude" => 116.3883,
  "latitude" => 39.9289,
  "continent_code" => "AS",
  "city_name" => "Beijing",
  "country_name" => "China",
  "country_code2" => "CN",
  "country_code3" => "CN",
  "region_code" => "11",
  "region_name" => "Beijing",
  "timezone" => “Asia/Shanghai",
  "location" => {
    "lon" => 116.3883,
    "lat" => 39.9289
  },
  "as_org" => "China Unicom Beijing Province Network",
  "asn" => 4808
}
```

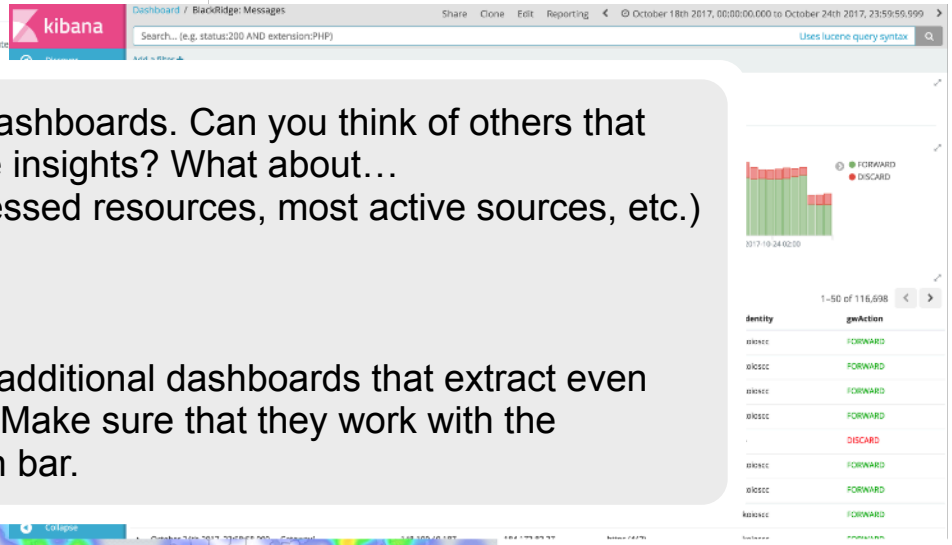
We probably do not need all of these fields, which means we are wasting storage space and performance.

CHALLENGE: Cleanup unnecessary fields prior to sending the document to Elasticsearch.

TIP: Take a look at the *fields* option of the *geoip* filter as well as the *mutate* filter. Additionally most filters support the *remove_field* option.

Even more Insights

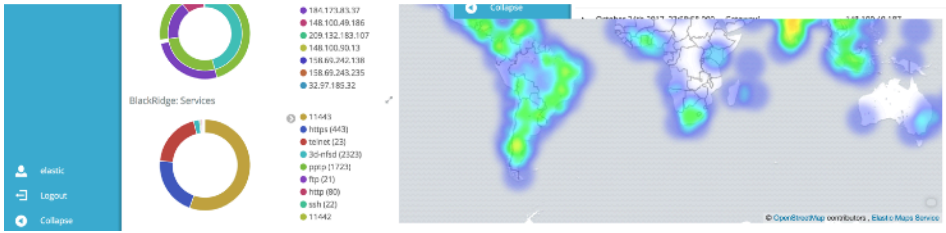




This tutorial includes three dashboards. Can you think of others that would add value and provide insights? What about…

- Top-N lists (most accessed resources, most active sources, etc.)
- Common connections
- ???

CHALLENGE: Add a few of additional dashboards that extract even more insights from the data. Make sure that they work with the markdown-based navigation bar.



elastic