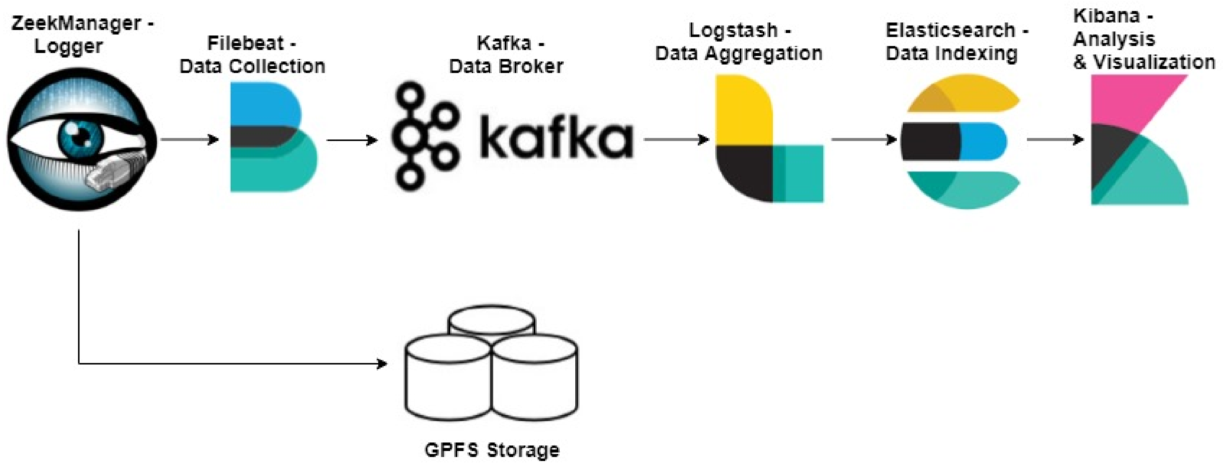
ELK

The Elastic Stack "also formally known as the ELK stack" is a free open source project software stack that is comprised of: Elasticsearch, Filebeat, Logstash, and Kibana. Elasticsearch is a data index and analytics engine. Filebeat is a lightweight log-data shipper. Logstash is a serverside data processing pipeline that can ingest data from multiple sources simultaneously, transform it to add data enrichment, and then send it to a 'stash' like Elasticsearch. Kibana lets users visualize data with charts and graphs from their indexed Elasticsearch data.

Along side the Elastic Stack, PULSAR utilizes a fault-tolerant, centralized data streaming platform called Kafka. This acts as a buffer between the IDS and Elasticsearch while also giving us the ability, in the future, to centrally collect different producer type data from things like syslog and netflow and being able to distribute that data along side the IDS data we are already collecting to Elasticsearch or other consumer type applications like Hadoop and Spark.

PULSAR utilizes the elastic stack along side Kafka as described in the image below:


ZeekManager - Logger
Filebeat - Data Collection
Kafka - Data Broker
kafka
Logstash - Data Aggregation
Elasticsearch - Data Indexing
Kibana - Analysis & Visualization
GPFS Storage

Filebeat - Data is collected from the ZeekManager and is given special type values based off their protocols before being sent to the kafka data-broker cluster. (ssh gets a type value of ssh, http logs get a type value of http etc.)
Since we have 36 different log types with unique filtering needs across many, these types are used later on in the pipeline by logstash to dynamically filter and enrich data as needed.

Kafka - Data is stored in what are known as kafka topics. These topics are broken down into kafka partitions across multiple drives and multiple servers to give read/write parallelism for higher throughput speeds and redundancy. Logs entering a topic are given an offset number that is used to tell the consumer application, in this case logstash, where it is in the data stream. That means if a consumer shuts down for any reason, it will resume where it left off when returning to service. This avoids data loss and old data being sent twice.

Logstash - Data fields are created with CSV filtering and fields are assigned data type values as needed "integer, Boolean" while also adding new fields for geoip location data and DNS reverse lookup data from originating and responding hosts.

Elasticsearch - Data coming into Elasticsearch is enriched one last time before being indexed by the use of elasticsearch templates. This template assigns our IP address fields the IP data type, and our Geoip fields as special Geo_Point types. The Ip type gives us the ability to query data based off subnet, while Geo-Point gives us the ability to create easy to make geographical map dashboards with a single field.

Elasticsearch Configuration

Install Elasticsearch

install the java dependency and perform a checksum against the SHA of the downloaded rpm (which should output elasticsearch-6.3.0.rpm: OK) then install

```
yum install java-1.8.0-openjdk
wget https://artifacts.elastic.co/downloads/elasticsearch/elasticsearch-
6.3.0.rpm
wget https://artifacts.elastic.co/downloads/elasticsearch/elasticsearch-
6.3.0.rpm.sha512
shasum -a 512 -c elasticsearch-6.3.0.rpm.sha512
sudo rpm -i elasticsearch-6.3.0.rpm
```

Cluster configuration

Elasticseach is deployed as a three node cluster. All three nodes operate as master eligible data nodes and follows a Nodes/2+1 quorum majority to avoid split brain situations.
Configuration file is location at **/etc/elasticsearch/elasticsearch.yml**, edit this accordingly across all clustered nodes.
Environment variables are used to keep the configuration files easily manageable in puppet.

```
##/etc/elasticsearch.yml

#name of cluster
cluster.name: elastic-cluster

#environment variable to print hostname
node.name: ${HOSTNAME}

#nodes are set as master/data by default but listed here for ease of
reference
node.master: true
node.data: true

#locations where elasticsearch will use for primary/replica shard
allocations. Separated by a comma if more than one location
path.data: /data/00/elasticsearch,/data/01/elasticsearch,/data/02
/elasticsearch,/data/03/elasticsearch,/data/04/elasticsearch,/data/05
/elasticsearch

#Path for log information to be stored
path.logs: /var/log/elasticsearch

#special value to print any site-local address of the node example:
(192.168.x.x) if more than one site-local interface you can directly
specify one with _[networkInterface]_
network.host: _site_

#List of nodes in the cluster
discovery.zen.ping.unicast.hosts: ["elastic-00", "elastic-01", "elastic-
02"]

#Prevent "split brain" situations by configuring the majority of nodes
needed to elect a new master (total number of master-eligible nodes / 2 +
1):
discovery.zen.minimum_master_nodes: 2
```

Edit the java heap space located in the config file **/etc/elasticsearch/jvm.options**
Maximum memory that can be allocated for heap is 32gb (recommended to stay between 26gb-32gb)

```
#Xms represents the initial size of total heap space
#Xmx represents the maximum size of total heap space

-Xms26gb
-Xmx26gb
```

Start cluster

Once jvm.option and elasticsearch.yml are configured start each node one by one.

```
systemctl start elasticsearch
```

You can use the CAT api to verify that the nodes are joining the cluster while also listing some useful master roll/resource usage information

```
curl elastic-00:9200/_cat/nodes?v

ip           heap.percent ram.percent cpu load_1m load_5m load_15m node.
role master name
192.168.x.x            54          60   2    1.19    1.10     1.16
mdi       -      elastic-02
192.168.x.x            69          55   2    0.70    1.07     1.14
mdi       *      elastic-00
192.168.x.x            45          62   6    0.37    0.97     1.21
mdi       -      elastic-01
```

Create Index Template

We need to specify some special values for both IP and Geoip fields so that we can utilize them in Kibana for dashboards.
Here's a template in it's entirety.

```
{
    "order" : 0,
    "version" : 001,
    "index_patterns" : [
      "IndexName*"
    ],
    "settings" : {
      "index" : {
        "refresh_interval" : "5s"
      }
    },
    "mappings" : {
      "doc" : {
        "dynamic_templates" : [
          {
            "message_field" : {
              "path_match" : "message",
              "match_mapping_type" : "string",
              "mapping" : {
                "type" : "text",
                "norms" : false
              }
            }
          },
          {
            "string_fields" : {
              "match" : "*",
              "match_mapping_type" : "string",
              "mapping" : {
                "type" : "text",
                "norms" : false,
                "fields" : {
                  "keyword" : {
                    "type" : "keyword",
                    "ignore_above" : 256
                  }
                }
              }
            }
          }
        ],
        "properties" : {
          "@timestamp" : {
            "type" : "date"
          },
          "@version" : {
            "type" : "keyword"
          },
          "id.orig_h" : {
            "type" : "ip"
          },
          "id.resp_h" : {
            "type" : "ip"
          },
          "resp_geoip" : {
            "dynamic" : true,
            "properties" : {
              "ip" : {
                "type" : "ip"
              },
              "location" : {
                "type" : "geo_point"
              },
              "latitude" : {
                "type" : "half_float"
              },
              "longitude" : {
                "type" : "half_float"
              }
            }
          },
      "orig_geoip" : {
            "dynamic" : true,
            "properties" : {
              "ip" : {
                "type" : "ip"
              },
              "location" : {
                "type" : "geo_point"
              },
              "latitude" : {
                "type" : "half_float"
              },
              "longitude" : {
                "type" : "half_float"
              }
            }
          }
        }
      }
    },
    "aliases" : { }
  }
```

Only thing to note to start is your index pattern. This is the name of the index you wish this template to apply to. If you are creating new dynamic indices daily based off date, example: IndexName-%{+YYYY.MM.dd}, you will need to specify an asterisk * as shown below.
"version" is an arbitrary number and can be used to keep track of new template changes/iterations.

```
      "order" : 0,
      "version" : 001,
      "index_patterns" : [
        "IndexName*"
```

This portion specifies that we need the two fields (id.orig_h, id.resp.h) that have IP address data in them assigned the special IP type value.

```
          "properties" : {
            "@timestamp" : {
              "type" : "date"
            },
            "@version" : {
              "type" : "keyword"
            },
            "id.orig_h" : {
              "type" : "ip"
            },
            "id.resp_h" : {
              "type" : "ip"
            },
```

This gives the child fields under the two parent Geoip data fields (resp_geoip, orig_geoip) the correct data types and assigns the "location" child field as the special "geo_point" value that we need later for Kibana dashboards.
(These child data fields are automatically created by logstash when using the Geoip filter)

```
            "resp_geoip" : {
              "dynamic" : true,
              "properties" : {
                "ip" : {
                  "type" : "ip"
                },
                "location" : {
                  "type" : "geo_point"
                },
                "latitude" : {
                  "type" : "half_float"
                },
                "longitude" : {
                  "type" : "half_float"
                }
              }
            },
        "orig_geoip" : {
              "dynamic" : true,
              "properties" : {
                "ip" : {
                  "type" : "ip"
                },
                "location" : {
                  "type" : "geo_point"
                },
                "latitude" : {
                  "type" : "half_float"
                },
                "longitude" : {
                  "type" : "half_float"
                }
```

Once the template is edited, save the file as template.json and upload it to Elasticsearch. TemplateName is an arbitrary name and can be whatever you like for your own reference.

```
#In same directory as your template.json file
curl -XPUT -H 'Content-Type: application/json' elastic-00:9200/_template
/TemplateName -d @template.json
```

Verify the template is now listed in the templates section using the CAT api.

```
curl elastic-00:9200/_cat/templates?v
```

The template (and other default templates) can be downloaded and edited if the need arises. once edited, just use the last two steps again to upload/verify. (changing the version number can help with keeping track of template changes)

```
#downloads the template "TemapleteName" and outputs it into a file named
template.json
curl elastic-00:9200/_template/TemplateName?pretty > template.json
```

Filebeat Configuration

Filebeat lives on the zeek IDS logger node and is configured to ship all currently generated logs in /usr/local/bro/logs/current/. Since there are 37 different log types generated by Zeek, we will focus on just two types (conn.log and weird.log) for examples.

Install filebeat

Download the filebeat rpm and install

```
wget https://artifacts.elastic.co/downloads/beats/filebeat/filebeat-7.1.0-
x86_64.rpm
sudo rpm -vi filebeat-7.1.0-x86_64.rpm
```

Filebeat configuration

Edit the **/etc/filebeat/filebeat.yml** to specify log files that need ingested and where to output those files to. In this case, we are sending all data to a kafka topic called "zeek-bulk." refer to the kafka cluster configuration on how to configure kafka and setup and create topics. If kafka is already configured to receive data, sending data to it with filebeat will automatically create a topic with default partition/replication settings.

here we are assigning every log with a special type based off log type (type: weird, type: ssh) this will be used later for special logstash filtering /data enrichment processing.

```
#marks each log as a weird type and ships log
- type: log
  paths:
    - /usr/local/bro/logs/current/weird.log
  fields:
    type: weird

#marks each log as a conn type and ships the log
- type: log
  paths:
    - /usr/local/bro/logs/current/conn.log
  fields:
    type: x509

#Tells filebeat to start reading the files from the end
  tail_files: true

#sends data to kafka brokers (fails-over to next host if first is not
responsive) to the Zeek-bulk topic
#if the topic doesn't exist in kafka it will automatically be created with
default topic settings
output.kafka:
  hosts: ["kafka-00:9092", "kafka-01:9092"]
  topic: zeek-bulk
  codec.json:
    pretty: false
```

Start filebeat

Start filebeat, starting filebeat in the foreground is a simple way to troubleshoot if you're having syntax issues.

```
systemctl start filebeat


#starts filebeat in foreground with active debugging information
filebeat -e -c /etc/filebeat/filebeat.yml
```

Confirgure logstash

Install logstash

Logstash runs on the kafka data broker nodes directly

Install the java dependency, downlaod and install the logstash rpm

```
yum install java-1.8.0-openjdk
wget https://artifacts.elastic.co/downloads/logstash/logstash-6.8.1.rpm
rpm -i logstash-6.8.1.rpm
```

Configure Logstash

Logstash config files are created and placed into the **/etc/logstash/conf.d/** directory. Logstash here is broken up into three config groups, input.conf, output.conf, and logtype.conf.

Create a file named input.conf, this will ingests data from a specified source, in this case a kafka topic.
This calls out the kafka instance running on port 9092 on the kafka-00 server and reads from the topic "zeek-bulk" while expecting data in json format.

```
input {
    kafka {
        bootstrap_servers => "kafka-00:9092"
        topics => "zeek-bulk"
        codec => json
   }
}
```

Once data is ingested, it needs to be filtered and enriched. Using the ("**type: conn", "type:weird**") that we assigned via Filebeat, we can apply spacial filtering to each log as needed.
Create a file named conn-logs.conf

```
#/etc/logstash/conf.d/conn-logs.conf

filter {
  if [fields][type] == "conn" {
    csv {
        separator => "        "
        columns => [ "worker", "ts", "uid", "id.orig_h", "id.orig_p", "id.
resp_h",
                     "id.resp_p", "proto", "service", "duration",
"orig_bytes",
                         "resp_bytes", "conn_state", "local_orig",
"local_resp",
                         "missed_bytes", "history", "orig_pkts",
"orig_ip_bytes",
                         "resp_pkts", "resp_ip_bytes", "tunnel_parents",
                         "orig_l1_addr", "resp_l2_addr" ]
    }

    date {
        match => [ "ts", "UNIX" ]
        target => "@timestamp"
    }
    geoip {
        source => "id.orig_h"
        target => "orig_geoip"
```

```
      }
      geoip {
          source => "id.resp_h"
          target => "resp_geoip"
      }
      mutate {
          add_field => { "id.orig_hostname" => "%{id.orig_h}" }
          add_field => { "id.resp_hostname" => "%{id.resp_h}" }
              remove_field => [ "agent", "log", "host", "ecs", "version" ]
              remove_field => [ "[_source][tags]" ]
          remove_field => [ "message" ]
          rename => [ "fields", "source" ]
          convert => {
              "orig_bytes" => "integer"
              "resp_bytes" => "integer"
              "orig_pkts" => "integer"
              "resp_pkts" => "integer"
              "orig_ip_bytes" => "integer"
              "resp_ip_bytes" => "integer"
              "missed_bytes" => "integer"
          }
      }
      dns {
          reverse => [ "id.resp_hostname", "id.orig_hostname" ]
          action => "replace"
          nameserver => [ "192.168.x.x" ]
          hit_cache_size => 10240
          hit_cache_ttl => 900
          failed_cache_size => 10240
          failed_cache_ttl => 900
      }
    }
  }
```

This is where our Filebeat "type:" will come into play. We specify that this filter will only run on logs marked as type "conn." This is important because conn logs have drastically different data and fields that our other log type/s don't have.
Our data is in a "CSV" tab delimited format so this takes each delimited piece of data and assigns it to a specific field.

```
filter {
  if [fields][type] == "conn" {
    csv {
        separator => "        "
        columns => [ "worker", "ts", "uid", "id.orig_h", "id.orig_p", "id.
resp_h",
                     "id.resp_p", "proto", "service", "duration",
"orig_bytes",
                         "resp_bytes", "conn_state", "local_orig",
"local_resp",
                         "missed_bytes", "history", "orig_pkts",
"orig_ip_bytes",
                         "resp_pkts", "resp_ip_bytes", "tunnel_parents",
                         "orig_l1_addr", "resp_l2_addr" ]
    }
```

This takes our ts field, that has a unix epoch timestamp, and converts it to a human readable form and places this newly transformed data into the @timestamp field.

```
    date {
        match => [ "ts", "UNIX" ]
        target => "@timestamp"
    }
```

This is where we generate our Geoip data for each field containing an IP address. Since we don't want to replace our main IP fields, we output this generated data to new fields (orig_geoip, resp_geoip)

```
    geoip {
        source => "id.orig_h"
        target => "orig_geoip"
    }
    geoip {
        source => "id.resp_h"
        target => "resp_geoip"
    }
```

Here we are using the mutate filter to do a number of different things to our data.
First we add new fields with duplicated data in them from our (id.resp_h id.orig_h) fields that have IP addresses them there. These fields will run through a DNS filter in the next step.
We then remove added metadata and uneeded fields generated automatically by logstash
We rename the "fields" field as source. When we use the (type: conn) in filebeats it's actually a child field of the parent field "fields." This just helps compered this is kibana. So instead of log being marked as "fields.type: conn" it's marked as "source.type:conn"
since we have data that we'd like to be able to add together and manipulate as numbers we need to convert the needed fields into integers. (by default logstash ships everything as strings)

```
    mutate {
        add_field => { "id.orig_hostname" => "%{id.orig_h}" }
        add_field => { "id.resp_hostname" => "%{id.resp_h}" }
            remove_field => [ "agent", "log", "host", "ecs", "version" ]
            remove_field => [ "[_source][tags]" ]
        remove_field => [ "message" ]
        rename => [ "fields", "source" ]
        convert => {
            "orig_bytes" => "integer"
            "resp_bytes" => "integer"
            "orig_pkts" => "integer"
            "resp_pkts" => "integer"
            "orig_ip_bytes" => "integer"
            "resp_ip_bytes" => "integer"
            "missed_bytes" => "integer"
        }
    }
```

Here is where we reverse lookup our IP data fields using the DNS filter.
We take our two newly crated fields and do a reverse lookup on them
You can list multiple nameserver here if desired, seperated by a comma
We have two cache options for successful/failed DNS queries. (how large the cache is and how long to keep successful/failed queries in cache before deleting)

```
    dns {
        reverse => [ "id.resp_hostname", "id.orig_hostname" ]
        action => "replace"
        nameserver => [ "192.168.x.x" ]
        hit_cache_size => 10240
        hit_cache_ttl => 900
        failed_cache_size => 10240
        failed_cache_ttl => 900
    }
```

Now create a file called output.conf
List out your elasticsearch hosts, and which index to store the data in. (The index will be automatically created if one does not exist)

```
output {
    elasticsearch {
        hosts => ["elastic-00:9200", "elastic-01:9200", "elastic-02:9200"]
        index => "zeek-logs"
    }
}
```

Start Logstash

Start the logstash and verify data is being sent to elasticsearch.

```
Systemctl start logstash


#uses to cat api to view index status, Verify the index is created and doc.
count is going up.
curl http://pulsar-es00:9200/_cat/indices?v


health status index     uuid                   pri rep docs.count docs.
deleted store.size pri.store.size
green  open   zeek-logs vtZLwY3CS4ic7UV2V96eSg   5   1
XX           0       XXgb           XXgb
green  open   .kibana   K1eLrK04RCyWC9L_g6yHIQ   1   1
4            0    154.9kb         77.4kb
```

Kibana configuration

Install kibana

```
wget https://artifacts.elastic.co/downloads/kibana/kibana-7.2.0-x86_64.rpm
sudo rpm -i kibana-7.2.0-x86_64.rpm
```

Configure kibana

Kibana needs very little to get started. Add an elasticseach node address and specify Kibana's host server IP.

```
#/etc/kibana/kibana.yml

#IP of machine running Elasticsearch
elasticsearch.url: "http://192.168.x.x:9200"
server.port: 5601
#IP of the kibana machine
server.host: "192.168.x.x" # IP of the kibana machine
```

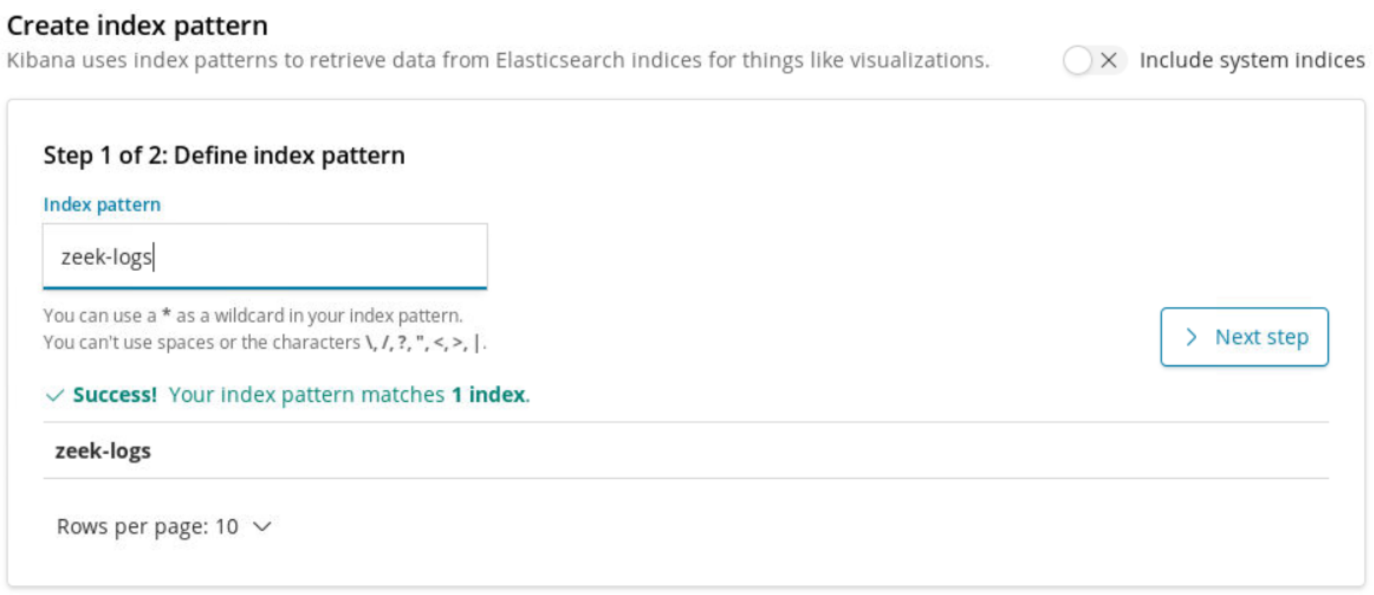
Start and navigate to web page

```
systemctl start kibana

#use browser of choice to navigate to web page
192.168.x.x:5601
```

Create index pattern

Opening kibana for the first time should prompt you to create an index pattern, if not navigate to management > index patterns and create one from there.
In the index pattern box enter your index name, in this case zeek-logs.


Create index pattern

Kibana uses index patterns to retrieve data from Elasticsearch indices for things like visualizations.

Include system indices

Step 1 of 2: Define index pattern

Index pattern

zeek-logs

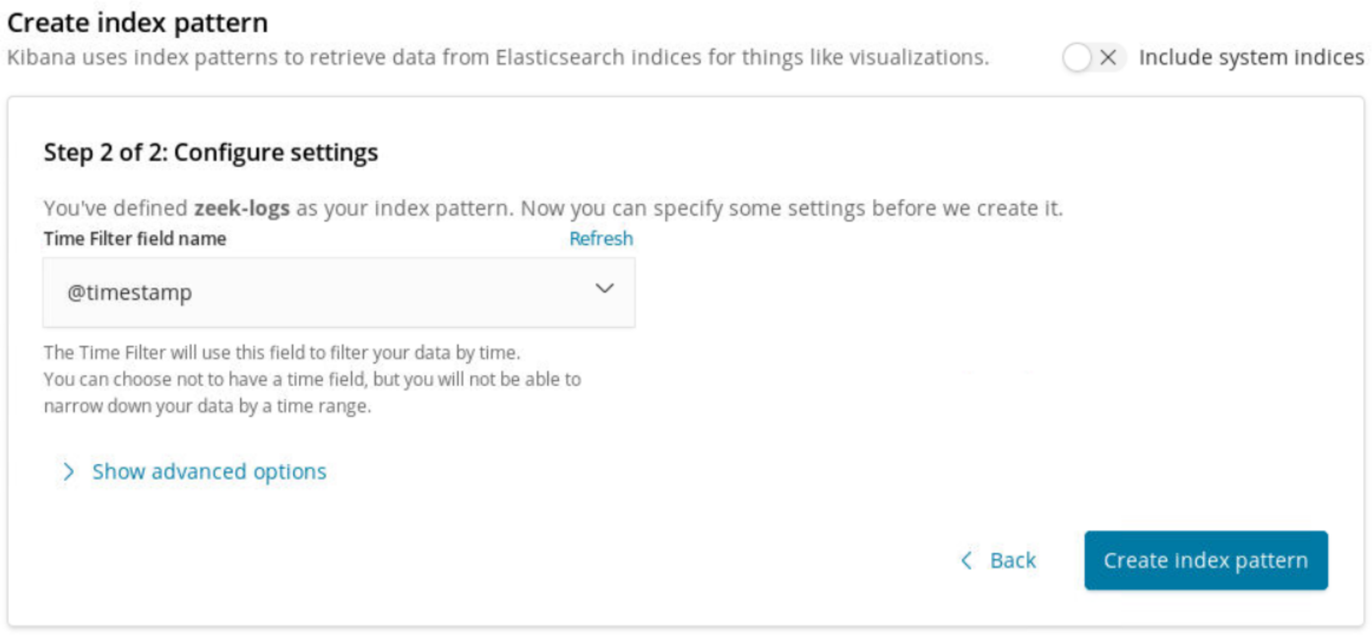
You can use a * as a wildcard in your index pattern.
You can't use spaces or the characters \, /, ?, ", <, >, | .

Success! Your index pattern matches 1 index.

zeek-logs

Rows per page: 10

Next select @timestamp in the drop-down time filter field and click "create index pattern"


Create index pattern

Kibana uses index patterns to retrieve data from Elasticsearch indices for things like visualizations.

Include system indices

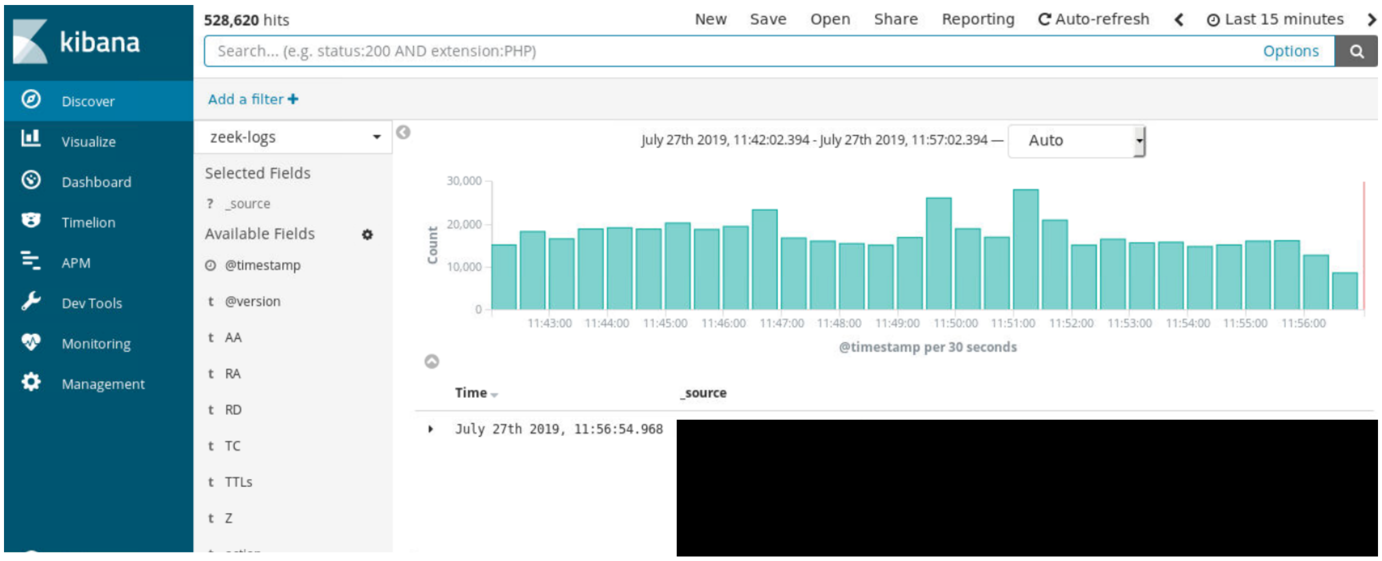
Step 2 of 2: Configure settings

You've defined **zeek-logs** as your index pattern. Now you can specify some settings before we create it.

Time Filter field name: @timestamp

The Time Filter will use this field to filter your data by time.
You can choose not to have a time field, but you will not be able to narrow down your data by a time range.

You can now navigate to "Discover" at the top left and view your index data.



528,620 hits

Search... (e.g. status:200 AND extension:PHP)

zeek-logs

Selected Fields
? _source

Available Fields
@timestamp
t @version
t AA
t RA
t RD
t TC
t TTLs
t Z

July 27th 2019, 11:42:02.394 - July 27th 2019, 11:57:02.394 — Auto

Time	_source
July 27th 2019, 11:56:54.968	[redacted]