Explore Centralized Logging

In this lesson, we will explore centralized logging through Elasticsearch, Fluentd, and Kibana.

WE'LL COVER THE FOLLOWING

Elasticsearch is probably the most commonly used in-memory database, At least if we narrow the scope to self-hosted databases. It is designed for many other scenarios, and it can be used to store (almost) any type of data. As such, it is almost perfect for storing logs, which could come in many different formats. Given its flexibility, some use it for metrics as well, and **Elasticsearch** competes with `Prometheus`. We'll leave metrics aside for now and focus only on logs.

EFK stack

The EFK (**Elasticsearch**, **Fluentd**, and **Kibana**) stack consists of three components. Data is stored in **Elasticsearch**. Logs are collected, transformed, and pushed to the DB by **Fluentd**, and **Kibana** is used as UI through which we can explore data stored in **Elasticsearch**. If you are used to ELK (**Logstash** instead of **Fluentd**), the setup that follows should be familiar.

The first component we'll install is **Elasticsearch**. Without it, **Fluentd** would not have a destination to ship logs, and **Kibana** would not have a source of

data.

Elasticsearch

As you might have guessed, we'll continue using `Helm` and, fortunately, Elasticsearch Chart is already available in the stable channel. I'm confident that you know how to find the chart and explore all the values you can use. So, we'll jump straight into the values I prepared. They are the bare minimum and contain only the `resources`.

```
cat logging/es-values.yml
```

```
client:
  resources:
    limits:
      cpu: 1
      memory: 1500Mi
    requests:
      cpu: 25m
      memory: 750Mi
master:
  resources:
    limits:
      cpu: 1
      memory: 1500Mi
    requests:
      cpu: 25m
      memory: 750Mi
data:
  resources:
    limits:
      cpu: 1
      memory: 3Gi
    requests:
      cpu: 100m
      memory: 1500Mi
```

As you can see, there are three sections (`client`, `master`, and `data`) that correspond with **ElasticSearch** components that will be installed. All we're doing is setting up resource requests and limits, and leaving the rest to the Chart's default values.

> Before we proceed, please note that you should NOT use those values in production. You should know by now that they differ from one case to another and that you should adjust resources depending on the actual usage that you can retrieve from tools like `kubectl top`, `Prometheus`, and others.

Let's install **Elasticsearch**.

```
kubectl create namespace logging

helm upgrade -i elasticsearch \
    stable/elasticsearch \
    --version 1.32.1 \
    --namespace logging \
    --values logging/es-values.yml

kubectl -n logging \
  rollout status \
  deployment elasticsearch-client
```

It might take a while until all the resources are created. On top of that, if you're using GKE, new nodes might need to be created to accommodate requested resources. Be patient.

Fluentd

Now that **Elasticsearch** is rolled out, we can turn our attention to the second component in the EFK stack. We'll install **Fluentd**. Just as **Elasticsearch**, **Fluentd** is also available in `Helm`'s stable channel.

```
helm upgrade -i fluentd \
    stable/fluentd-elasticsearch \
    --version 2.0.7 \
    --namespace logging \
    --values logging/fluentd-values.yml

kubectl -n logging \
    rollout status \
    ds fluentd-fluentd-elasticsearch
```

There's not much to say about **Fluentd**. It is running as **DaemonSet** and, as

the name of the Chart suggests, it is already preconfigured to work with

Elasticsearch. I did not even bother showing you the contents of the values file `logging/fluentd-values.yml` since it contains only the resources.

To be on the safe side, we'll check **Fluentd**'s logs to confirm that it managed to connect to **Elasticsearch**.

```
kubectl -n logging logs \
    -l app.kubernetes.io/instance=fluentd
```

The **output**, limited to the messages, is as follows.

```
... Connection opened to Elasticsearch cluster => {:host=>"elasticsearch-client", :port=>9200, :scheme=>"http"}
```

> ### A note to Docker For Desktop users
>
> You will likely see much more than the few log entries presented above. There will be a lot of warnings due to the differences in Docker For Desktop API when compared to other Kubernetes flavors. Feel free to ignore those warnings since they do not affect the examples we are about to explore and you are not going to use Docker For Desktop in production but only for practice and local development.

That was simple and beautiful. The only thing left is to install the **K** from EFK.

Kibana Chart

Let's take a look at the values file we'll use for the Kibana chart.

```
cat logging/kibana-values.yml
```

The output is as follows.

```
ingress:
  enabled: true
  hosts:
  - acme.com
env:
  ELASTICSEARCH_URL: http://elasticsearch-client:9200
```

```
resources:
  limits:
    cpu: 50m
    memory: 300Mi
  requests:
    cpu: 5m
    memory: 150Mi
```

Again, this is a relatively straightforward set of values. This time, we are specifying not only the resources but also the Ingress host, as well as the environment variable `ELASTICSEARCH_URL` which will tell **Kibana** where to find **Elasticsearch**. As you might have guessed, I did not know in advance what your host would be, so we'll need to overwrite `hosts` at runtime. But, before we do that, we need to define it.

```
KIBANA_ADDR=kibana.$LB_IP.nip.io
```

Off we go towards installing the last component in the EFK stack.

```
helm upgrade -i kibana \
    stable/kibana \
    --version  3.2.5 \
    --namespace logging \
    --set ingress.hosts="{$KIBANA_ADDR}" \
    --values logging/kibana-values.yml

kubectl -n logging \
    rollout status \
    deployment kibana
```

Confirm all EFK components are running

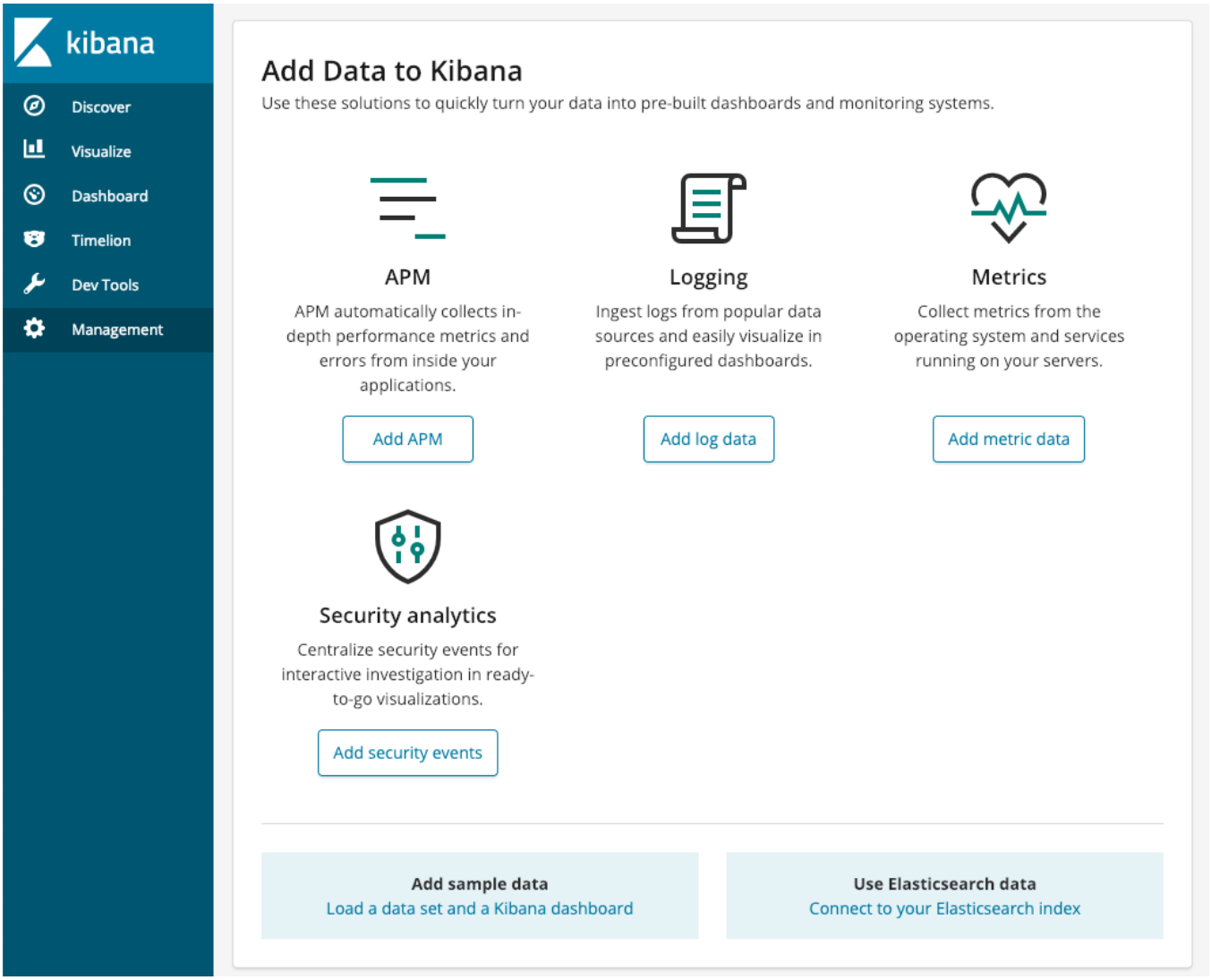
Now we can finally open **Kibana** and confirm that all three EFK components indeed work together and that they are fulfilling our centralized logging objectives.

```
open "http://$KIBANA_ADDR"
```

If you do not see **Kibana** just yet, wait for a few moments and refresh the screen.

You should see the *Welcome* screen. Ignore the offer to *try* their *sample data*

by clicking the link to *Explore on my own*. You'll be presented with a screen that allows you to add data.



Kibana's home screen

Elasticsearch Index

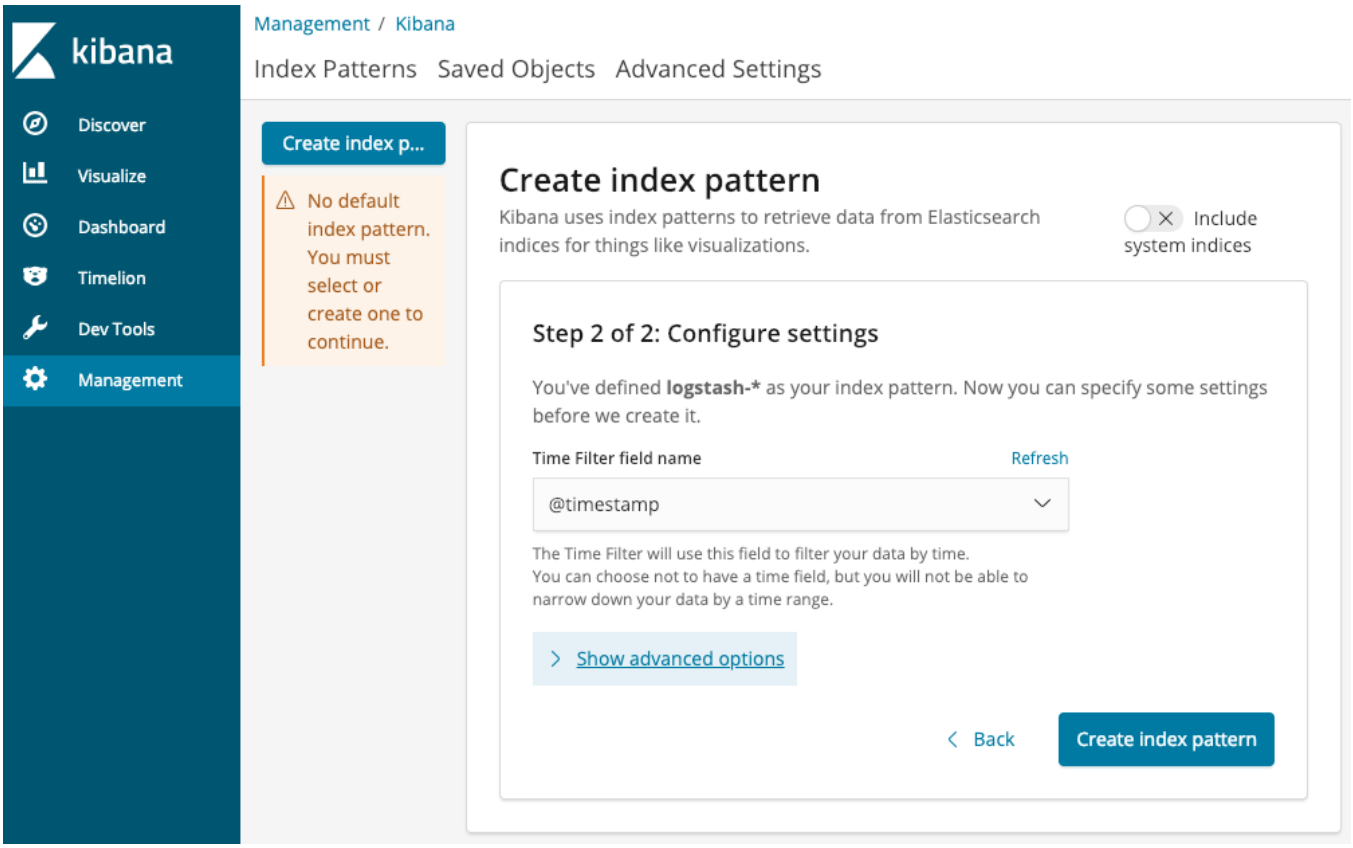
The first thing we need to do is create a new **Elasticsearch** index that will match the one created by **Fluentd**. The version we're running is already pushing data to **Elasticsearch**, and it's doing that by using the **LogStash** indexing pattern as a way to simplify things since that's what **Kibana** expects to see.

Click the *Management* item from the left-hand menu, followed by the *Index Patterns* link.

All the logs **Fluentd** is sending to **Elasticsearch** are indexes with the *logstash* prefix followed with the date. Since we want **Kibana** to retrieve all the logs, type `logstash-*` into the *Index pattern* field, and click the > *Next step* button.

Next, we need to specify which field contains timestamps. It is an easy choice.

Select *@timestamp* from the *Time Filter field name*, and click the *Create index pattern* button.

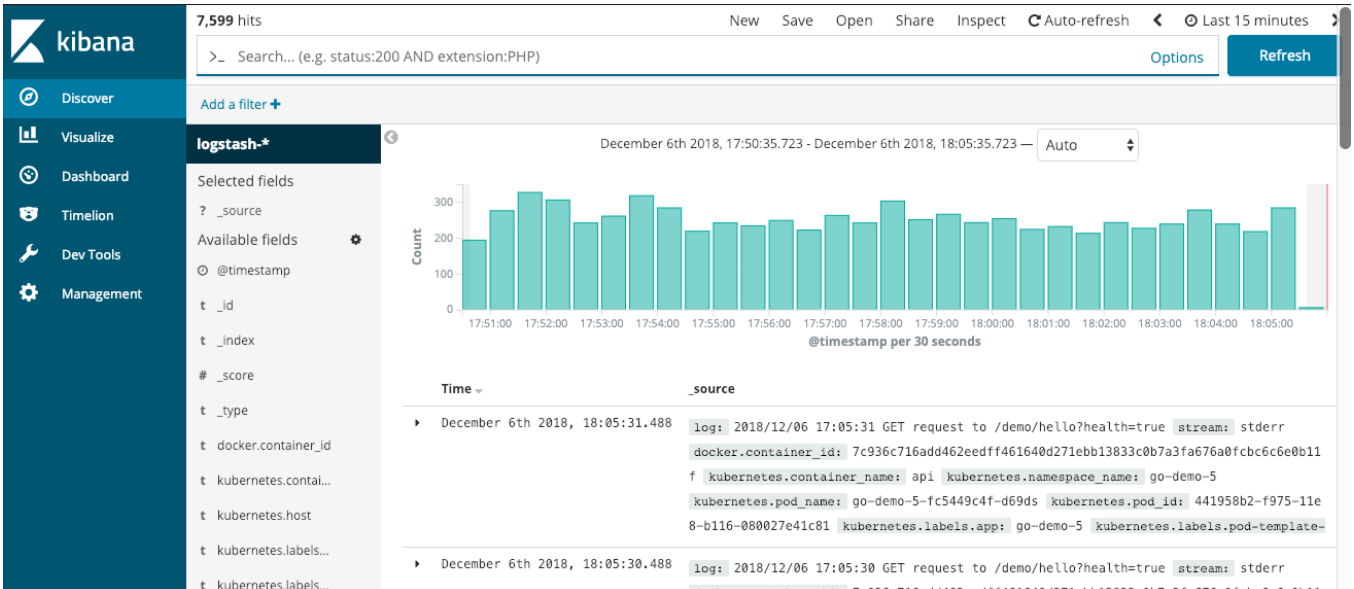


Kibana's Create index pattern screen

That's it. All we have to do now is wait for a few moments until the index is created, and explore the logs collected from across the whole cluster.

Please click the *Discover* item from the left-hand menu.

What you see in front of you are all the logs generated in the last fifteen minutes (can be extended to any period). The list of fields is available on the left-hand side. There's a silly (and useless) graph on the top and the logs themselves are in the main body of the screen.



Kibana's Discover screen

Few basic operations in Kibana

Just as with **Papertrail**, we won't go into all the options available in **Kibana**. I trust you can figure them out yourself. We'll just go through a few basic operations in case this is your first contact with **Kibana**.

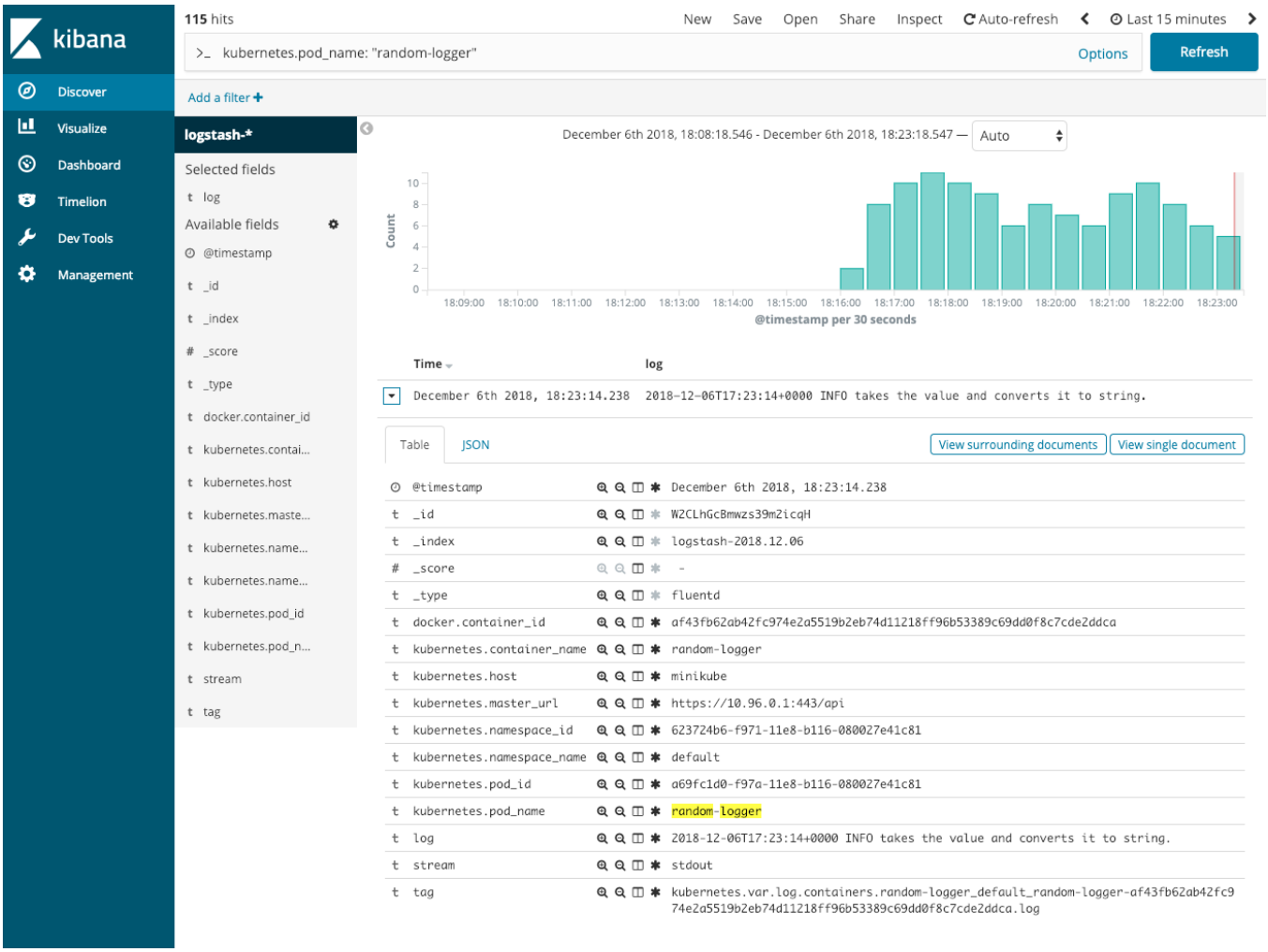
Our scenario is the same as before. We'll try to find all the log entries generated from the *random-logger* application.

Please type `kubernetes.pod_name: "random-logger"` into the *Search* field and click the *Refresh* (or *Update*) button located on the right.

More often than not, we want to customize the fields presented by default. For example, it would be more useful to see the log entry only, instead of the full source.

Click the *Add* button next to the *log* field, and it will replace the default *_source* column.

If you'd like to see an entry with all the fields, please expand one by clicking the arrow on the left side of the row.



Kibana's Discover screen with filtered entries

I'll leave you to explore the rest of **Kibana** on your own. But, before you do, there's a word of warning. Do not get fooled by all the flashy options. If all we're having are logs, there's probably no point creating visualizations, dashboards, timelines, and other nice-looking, but useless things we can do with logs. Those might be useful with metrics, but we do not have any. For now, they are in `Prometheus`. Later on, we'll discuss the option of pushing metrics to **Elasticsearch** instead of pulling them from `Prometheus`.

Now, take your time and see what else you can do in **Kibana**, at least within the Discover screen.

Delete the EFK stack

We're done with the EFK stack and, given that we did not yet make a decision which solution to use, we'll purge it from the system. Later on, if you do choose EFK, you should not have any trouble creating it in your "real" cluster.

```
helm delete kibana \
    --namespace logging


helm delete fluentd \
    --namespace logging

helm delete elasticsearch \
    --namespace logging

kubectl -n logging \
    delete pvc \
    -l release=elasticsearch,component=data

kubectl -n logging \
    delete pvc \
    -l release=elasticsearch,component=master
```

In the next lesson, we will see how to use **Elasticsearch** for storing metrics.