

Chapter 14: Monitoring the KVM Virtualization Platform

When you move away from an environment that only has a couple of objects to manage (for example, KVM hosts) to an environment that has hundreds of objects to manage, you start asking yourself very important questions. One of the most prominent questions is, *How am I going to monitor my hundreds of objects without doing a lot of manual work and with some GUI reporting options?* And the answer to that question is the **Elasticsearch, Logstash, Kibana (ELK)** stack. In this chapter, we'll see what these software solutions can do for you and your KVM-based environment.

Behind those cryptic names are technologies that are here to solve a lot of problems you might have when running more than one server. Although you can run the ELK stack to monitor one service, it makes no sense to do so. The advice and solutions provided in this chapter are applicable to all projects involving multiple devices and servers, not only those running on

KVM but, in essence, anything that is capable of producing any kind of logging. We will start with the basics of how to monitor KVM as a virtualization platform in general. Then, we'll move on to the ELK stack, including its building blocks and installation, before moving on to its advanced configuration and customization.

In this chapter, we will cover the following topics:

- Monitoring the KVM virtualization platform
- Introduction to the open source ELK solution
- Setting up and integrating the ELK stack
- Configuring the data collector and aggregator
- Creating custom utilization reports
- Let's get started!

Monitoring the KVM virtualization platform

When we talk about running a system that is performing any kind of processing, we quickly come to the problem of monitoring and making sure that our system runs inside a given set of parameters.

When we create a system that is running a workload, it will inevitably produce some kind of data

on everything that is happening. This data can be almost infinite in its scope – a server that is just online, without a single *useful* task running will create some kind of log or service data, such as the amount of used memory, services that are starting or stopping, the amount of disk space left, devices that are connecting and disconnecting, and more.

When we start running any useful task, the logs will only get larger.

Having a good and verbose log means that we can find what is going on at this instant with the system; is it running correctly and do we need to do something to make it run better? If something unexpected happens, logs can help us determine what is actually wrong and point us in the direction of the solution. Correctly configured logs can even help us spot errors before they start to create problems.

Suppose you have a system that is getting slower and slower week after week. Let's further suppose that our problem is with the memory allocation of an app we installed on the system. But let's also suppose that this memory allocation is not constant, and instead varies with the number of users using the system. If you take a look at any point in time, you may notice the number of

users and memory allocated. But if you just take measurements at different times, you will have a hard time understanding what kind of correlation there is between the memory and the number of users – will the amount of memory allocated be linear to the number of users or will it behave exponentially? If we can see that 100 users are using 100 MB of memory, does that mean that 1,000 users will use 1,000 MB?

But let's suppose that we are logging the amount of memory and the number of users at equally spaced intervals.

We are not doing anything complicated; every couple of seconds, we are writing down the time of the measurement, the amount of memory allocated, and the number of users using the system. We are creating something called a dataset, consisting of **data points**. Using data points is no different than what we did in the preceding example, but once we have a dataset, we can do trend analysis. Basically, instead of looking at a slice of the problem, we can analyze different time segments and compare the number of users and what the amount of memory they were using actually was. That will give us important information about how our system is actually using our

memory and at what point we had a problem, even if we don't have a problem right now.

This approach can even help us find and troubleshoot problems that are non-obvious, such as a backup that is taking too long to finish once a month and works normally the rest of the time. This kind of capability that enables us to spot trends and analyze data and system performance is what logging is all about.

Put simply, any kind of monitoring boils down to two things: collecting data from the thing we are trying to monitor and analyzing that data.

Monitoring can be either online or offline.

Online monitoring is useful when we are trying to create some sort of alerting system or when we are trying to establish the self-correcting system that will be able to respond to changes in the process. Then, we can either try to correct problems or shut down or restart the system. Online monitoring is usually used by the operations team in order to make sure that everything is running smoothly and that the problems the system may have are logged.

Offline monitoring is much more complicated. Offline monitoring enables us to gather all the data into logs, analyze these logs later, and ex-

trapolate trends and figure out what can be done to the system to make it better. But the fact of the matter is that it's always *delayed* in terms of real-time activity since the offline methodology requires us to *download* and then *analyze* the logs. That's why we prefer real-time log ingestion, which is something that needs to be done online. That's why learning about the ELK stack is so important.

By fitting all these small pieces – real-time log ingestion, search, analytics, and reports –into one larger stack, ELK makes it easier for us to monitor our environment in real time. Let's learn how.

Introduction to the open source ELK solution

We mentioned previously that ELK stands for Elasticsearch, Logstash, and Kibana because these three applications or systems are the building blocks of a complete monitoring and reporting solution. Each part has its own purpose and functions it performs – Logstash gathers all the data into a consistent database, Elasticsearch is able to quickly go through all the data that Logstash stored, and Kibana is here to turn

search results into something that is both informational and visually appealing. Having said all this, ELK recently changed its name. Although it is still referred to as the ELK Stack, and almost the entirety of the internet will call it that, the ELK stack is now named the Elastic Stack, for the sole reason that, at the time of writing, there is another fourth component included in the stack. This component is called Beats, and it represents a significant addition to the whole system.

But let's start from the beginning and try to describe the whole system the way its creators describe it.

Elasticsearch

The first component that was created and that got traction in the community was Elasticsearch, created to be a flexible, scalable system for indexing and searching large datasets.

Elasticsearch was used for thousands of different purposes, including searching for specific content in documents, websites, or logs. Its main selling point and the reason a lot of people started using it is that it is both flexible and scalable, and at the same time extremely fast.

When we think of searching, we usually think about creating some kind of query and then

waiting for the database to give us back some form of answer. In complex searches, the problem is usually the waiting since it is exhausting having to tweak our queries and wait for them to produce results. Since a lot of modern data science relies on the concept of non-structured data, meaning that a lot of data that we need to search has no fixed structure, or no structure at all, creating a fast way to search inside this pool of data is a tough problem.

Imagine you need to find a certain book in a library. Also, imagine you do not have a database of all the books, authors, publishing information, and everything else that a normal library has; you are only allowed to search through all the books themselves.

Having a tool that is able to recognize patterns in those books and that can tell you the answer to questions such as *who wrote this book?* or *how many times is KVM mentioned in all the books that are longer than 200 pages?* is a really useful thing. This is what a good search solution does.

Being able to search for a machine that is running the Apache web server and has problems with a certain page requested by a certain IP address is essential if we want to quickly and effi-

ciently administer a cluster or a multitude of clusters of physical and virtual servers.

The same goes for system information when we are monitoring even a single point of data, such as memory allocation across hundreds of hosts. Even presenting that data is a problem and searching for it in real time is almost impossible without the right tool.

Elasticsearch does exactly that: it creates a way for us to quickly go through enormous amounts of barely structured data and then comes up with results that make sense. What makes Elasticsearch different is its ability to scale, which means you can use it to create search queries on your laptop, and later just run them on a multi-node instance that searches through a petabyte of data.

Elasticsearch is also fast, and this is not something that only saves time. Having the ability to get search results faster gives you a way to learn more about your data by creating and modifying queries and then understanding their results.

Since this is just a simple introduction to what ELK actually does, we will switch to the next component, Logstash, and come back to searching a bit later.

Logstash

Logstash has a simple purpose. It is designed to be able to digest any number of logs and events that generate data and store them for future use. After storing them, it can export them in multiple formats such as email, files, HTTP, and others.

What is important about how Logstash works is its versatility in accepting different input streams. It is not limited to using only logs; it can even accept things such as Twitter feeds.

Kibana

The last part of the old ELK stack is Kibana. If Logstash is storage and Elasticsearch is for computing, then Kibana is the output engine. Simply put, Kibana is a way to use the results of Elasticsearch queries to create visually impressive and highly customizable layouts. Although the output of Kibana is usually some kind of a dashboard, its output can be many things, depending on the user's ability to create new layouts and visualize data. Having said all this, don't be afraid – the internet offers at least a partial, if not full solution, to almost every imaginable scenario.

Next, what we will do is go through the basic installation of the ELK stack, show what it can do, point you in the right direction, and demonstrate one of the most popular *beats* – **metricbeat**.

Using the ELK stack is, in many ways, identical to *running* a server – what you need to do depends on what you actually want to accomplish; it takes only a couple of minutes to get the ELK stack running, but the real effort only starts there.

Of course, for us to fully understand how the ELK stack is used in a live environment, we need to deploy it and set it up first. We'll do that next.

Setting up and integrating the ELK stack

Thankfully, almost everything that we need to install is already prepared by the Elasticsearch team. Aside from Java, everything is nicely sorted and documented on their site.

The first thing you need to do is install Java – ELK depends on Java to run, so we need to have it installed. Java has two different install candidates: the official one from Oracle and the open source OpenJDK. Since we are trying to stay in the open source ecosystem, we'll install

OpenJDK. In this book, we are using CentOS 8 as our platform, so the **yum** package manager will be used extensively.

Let's start with the prerequisite packages. The only prerequisite package we need in order to install Java is the **java-11-OpenJDK-devel** package (substitute "11" with the current version of OpenJDK). So, here, we need to run the following command:

```
yum install java-11-openjdk-devel
```

After issuing that command, you should get a result like this:

```
[root@localhost yum.repos.d]# yum install java-11-openjdk-devel
Updating Subscription Management repositories.
Last metadata expiration check: 0:36:07 ago on Sat 27 Jul 2019 04:37:07 PM EDT.
Dependencies resolved.
=====
Package           Arch    Version                               Repository                               Size
=====
Installing:
java-11-openjdk-devel
x86_64 1:11.0.4.11-0.el8_0 rhel-8-for-x86_64-appstream-rpms 3.4 M
Installing dependencies:
javapackages-filesystem
noarch 5.3.0-1.module+el8+2447+6f56d9a6 rhel-8-for-x86_64-appstream-rpms 30 k
xorg-x11-fonts-Type1
noarch 7.5-19.el8 rhel-8-for-x86_64-appstream-rpms 522 k
copy-jdk-configs
noarch 3.7-1.el8 rhel-8-for-x86_64-appstream-rpms 27 k
ttmkfdir
x86_64 3.0.9-54.el8 rhel-8-for-x86_64-appstream-rpms 62 k
java-11-openjdk-headless
x86_64 1:11.0.4.11-0.el8_0 rhel-8-for-x86_64-appstream-rpms 39 M
tzdata-java
noarch 2019b-1.el8 rhel-8-for-x86_64-appstream-rpms 189 k
java-11-openjdk
x86_64 1:11.0.4.11-0.el8_0 rhel-8-for-x86_64-appstream-rpms 227 k
lksectp-tools
x86_64 1.0.18-3.el8 rhel-8-for-x86_64-baseos-rpms 100 k
Enabling module streams:
javapackages-runtime
201801

Transaction Summary
=====
Install 9 Packages

Total download size: 44 M
Installed size: 180 M
Is this ok [y/N]: y
```

Figure 14.1 – Installing one of the main prerequisites – Java

Once installed, we can verify whether the setup was successful and whether Java is working properly by running the following command:

```
java -version
```

This is the expected output:

```
root@localhost yum.repos.d]# java -version
penjdk version "11.0.4" 2019-07-16 LTS
penJDK Runtime Environment 18.9 (build 11.0.4+11-LTS)
penJDK 64-Bit Server VM 18.9 (build 11.0.4+11-LTS, mixed mode, sharing)
root@localhost yum.repos.d]#
```

Figure 14.2 – Checking Java's version

The output should be the current version of Java and no errors. Other than verifying whether Java works, this step is important in order to verify that the path to Java is correctly set – if you are running on some other distributions, you may have to set the path manually.

Now that java is installed and ready to go, we can continue with the installation of the ELK stack. The next step is to configure the install source for Elasticsearch and other services:

1. We need to create a file in **/etc/yum.repos.d/** named **elasticsearch.repo** that will contain all the information about our repository:

```
[Elasticsearch-7.x]
name=Elasticsearch repository for
7.x packages
baseurl=https://artifacts.elastic.co/packages/7.x/yum
```

```
gpgcheck=1
gpgkey=https://artifacts.elastic.co
/GPG-KEY-Elasticsearch
enabled=1
autorefresh=1
type=rpm-md
```

Save the file. The important thing here is that the repository is GPG-signed, so we need to import its key and apply it so that the packages can be verified when they're downloaded.

The files that you are going to install are not free software. Elasticsearch has two distinct free versions and a paid subscription model. What you are going to get using the files in this repository is the subscription-based install that is going to run in *basic* mode, which is free. At the time of writing, Elastic has four subscription models – one is open source, based on the Apache License 2.0, and free; the rest of them are closed source but offer additional functionalities. Currently, these subscriptions are named Basic, Gold, and Platinum. Basic is free, while the other models require a monthly paid subscription.

You will inevitably ask why you should choose open source over Basic, or vice versa since they are both free. While both of them have the same core, Basic is more advanced as it offers core security features and more things

that can be important in everyday use, especially if you are after Kibana visualizations.

2. Let's continue with the installation and import the necessary GPG key:

```
rpm --import  
https://artifacts.elastic.co/GPG-  
KEY-elasticsearch
```

3. Now, we are ready to do some housekeeping on the system side and grab all the changes in the repository system:

```
sudo yum clean all  
sudo yum makecache
```

If everything is okay, we can now install **elasticsearch** by running this command:

```
sudo yum install elasticsearch
```

Neither **elasticsearch** nor any of the other services are going to be started or enabled automatically. We must do this manually for each of them. Let's do that now.

4. The procedure to start and enable services is standard and is the same for all three services:

```
sudo systemctl daemon-reload  
sudo systemctl enable  
elasticsearch.service  
sudo systemctl start  
elasticsearch.service  
sudo systemctl status  
elasticsearch.service  
sudo yum install kibana
```

```
sudo systemctl status  
kibana.service  
sudo systemctl enable  
kibana.service  
sudo systemctl start kibana.service  
sudo yum install logstash  
sudo systemctl start  
logstash.service  
sudo systemctl enable  
logstash.service
```

The last thing to do is installing *beats*, which are services that are usually installed on the monitored servers, and which can be configured to create and send important metrics on the system. Let's do that now.

5. For the purpose of this demonstration, we will install them all, although we are not going to use all of them:

```
sudo yum install filebeat  
metricbeat packetbeat heartbeat-  
elastic auditbeat
```

After this, we should have a functional system. Let's have a quick overview.

Kibana and Elasticsearch are both running as web services, on different ports. We are going to interact with Kibana via the web browser (using the URLs **http://localhost:9200** and

http://localhost:5601) since this is where the visualization happens:

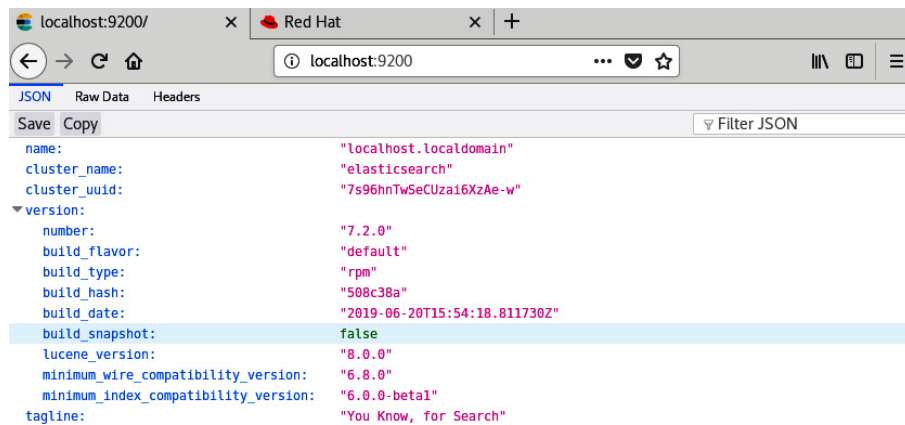


Figure 14.3 – Checking the Elasticsearch service

Now, we can connect to Kibana on port **5601**:

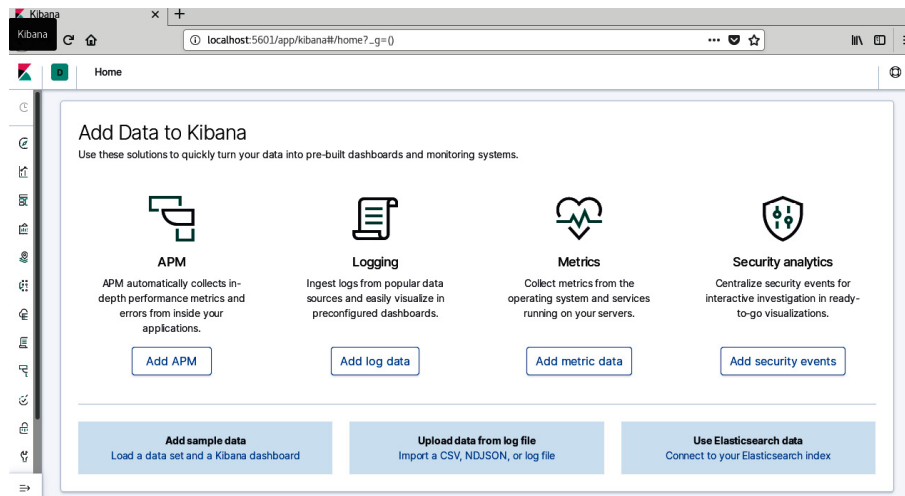


Figure 14.4 – Successful connection to Kibana

With that, the deployment process was finished successfully. Our logical next step would be to create a workflow. Let's do that now.

Workflow

In this section, we are going to establish a workflow – we are going to create logs and metrics

that are going to be ingested into Logstash, queried via Elasticsearch, and then visually represented in Kibana.

By default, Kibana runs on port **5601**, which can be changed in the configuration.

But what does this mean for me? What does this mean for KVM?

The biggest selling point for using Elastic stack is flexibility and ease of presentation. It doesn't matter if we are running one, 10, or 1,000 machines inside dozens of KVM hosts; we can treat them the same in production and establish a stable monitoring workflow. Using extremely simple scripts, we can create completely custom metrics and quickly display them, we can watch for trends, and we can even create a near-real-time monitoring system. All this, essentially for free.

Let's create a simple monitor that is going to dump system metrics for the host system that is running ELK. We've already installed Metricbeat, so the only thing left is to configure the service to send the data to Elasticsearch. Data is sent to Elasticsearch, not Logstash, and this is simply because of the way that the services interoperate. It is possible to send both to Logstash and

Elasticsearch, so we need to do a quick bit of explaining here.

Logstash is, by definition, a service that stores data that's sent to it. Elasticsearch searches that data and communicates with Logstash. If we send the data to Logstash, we are not doing anything wrong; we are just dumping data for later analysis. But sending to Elasticsearch gives us one more feature – we can send not only data but also information about the data in the form of templates.

On the other hand, Logstash has the ability to perform data transformation right after it receives it and before data is stored, so if we need to do things such as parse GeoIP information, change the names of hosts, and so on, we will probably use Logstash as our primary destination. Keeping that in mind, do not set Metricbeat so that it sends data both to Elasticsearch and Logstash; you will only get duplicate data stored in the database.

Using ELK is simple, and we've got this far into the installation without any real effort. When we start analyzing the data is when the real problems start. Even simple and perfectly formatted data that comes out of Metricbeat can be complex to visualize, especially if we are doing it for

the first time. Having premade templates both for Elasticsearch and Kibana saves a lot of time.

Take a look at the following screenshot:

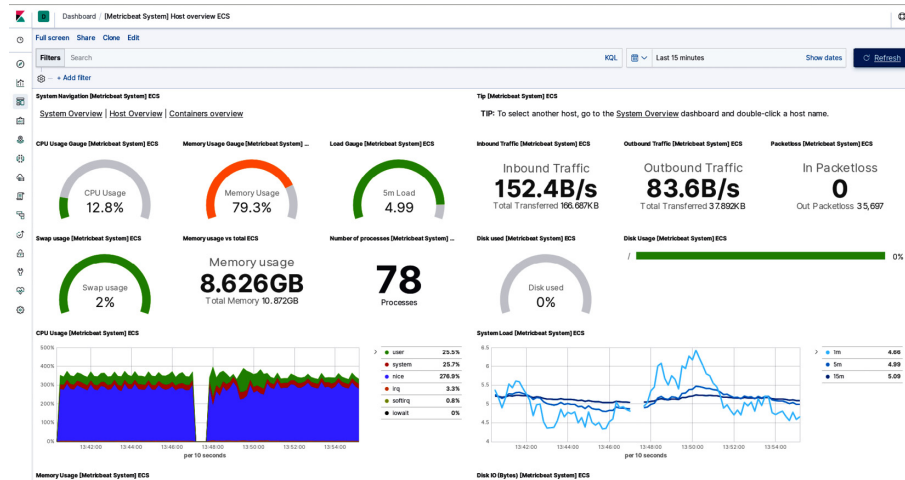


Figure 14.5 – Metricbeat dashboard

It takes no more than 10 minutes of setup to get a complete dashboard like this one. Let's go through this step by step.

We already have Metricbeat installed and only need to configure it, but before that, we need to configure Logstash. We only need to define one *pipeline*.

So, how can data be transformed?

Up until now, we did not go into details regarding how Logstash functions, but to create our first set of data, we need to know some of the inner workings of Logstash. Logstash uses a concept of a pipeline to define what happens to data

once it's received, and before that data is sent to Elasticsearch.

Each pipeline has two required, and one optional, elements:

- The input is always the first in the pipeline and is designed to receive data from the source.
- The output is the last element in the pipeline, and it outputs the data.
- The filter is an optional element and stands between the input and output in order to modify the data in accordance with the rules that we can define.

All these elements can be chosen from a list of plugins in order for us to create an optimal pipeline adjusted for a specific purpose. Let's go through this step by step.

What we need to do is just uncomment the one pipeline that is defined in the configuration file, located in the **/etc/logstash** folder.

The whole stack uses YAML as the standard for the configuration file structure, so every configuration file ends with the **.yaml** extension. This is important in order to understand that all the files that do not have this extension are here as either a sample or some kind of template for the

configuration; only files with the **.yaml** extension will get parsed.

To configure Logstash, just open **logstash.yaml** and uncomment all the lines that are related to the first pipeline, called **main**. We don't need to do anything else. The file itself is located in the **/etc/logstash** folder, and should look something like this after you make these changes:

```
#
# pipeline.id: main
#
# Set the number of workers that will, in parallel, execute the filters+outputs
# stage of the pipeline.
#
# This defaults to the number of the host's CPU cores.
#
# pipeline.workers: 2
#
# How many events to retrieve from inputs before sending to filters+workers
#
# pipeline.batch.size: 125
#
# How long to wait in milliseconds while polling for the next event
# before dispatching an undersized batch to filters+outputs
#
# pipeline.batch.delay: 50
#
# Force Logstash to exit during shutdown even if there are still inflight
# events in memory. By default, logstash will refuse to quit until all
# received events have been pushed to the outputs.
#
```

Figure 14.6 – The logstash.yaml file

The next thing we need to do is configure Metricbeat.

Configuring data collector and aggregator

In the previous steps, we managed to deploy Metricbeat. Now, we need to start the actual configuration. So, let's go through the configuration procedure, step by step:

1. Go to `/etc/metricbeat` and open

`metricbeat.yml`.

Uncomment the lines that define **elastic-search** as the target for Metricbeat. Now, we need to change one more thing. Find the line containing the following:

```
setup.dashboards.enabled: false
```

Change the preceding line to the following:

```
setup.dashboards.enabled: true
```

We need to do this to load dashboards so that we can use them.

2. The rest of the configuration is done from the command line. Metricbeat has a couple of commands that can be run, but the most important is the following one:

```
metricbeat setup
```

This command will go through the initial setup. This part of the setup is probably the most important thing in the whole initial configuration – pushing the dashboard templates to Kibana. These templates will enable you to get up and running in a couple of clicks, as opposed to learning how to do visualization and configuring it from scratch. You will have to do this eventually but for this example, we want to get things running as quickly as possible.

3. One more command that you need right now is the following one:

```
metricbeat modules list
```

This will give you a list of all the modules that Metricbeat already has prepared for different services. Go ahead and enable two of them, **logstash** and **kvm**:

```
metricbeat modules enable kvm  
metricbeat modules enable logstash
```

The **logstash** module is confusingly named since it is not intended to push data to Logstash; instead, its main purpose is to report the Logstash service and enable you to monitor it through Logstash. Sound confusing? Let's rephrase this: this module enables Logstash to monitor itself. Or to be more precise, it enables beats to monitor part of the Elastic stack.

The KVM module is a template that will enable you to gather different KVM-related metrics.

This should be it. As a precaution, type in the following command to check Metricbeat's configuration:

```
metricbeat test config
```

If the preceding command runs okay, start the Metricbeat service using the following command:

```
systemctl start metricbeat
```

You now have a running service that is gathering data on your host – the same one that is running

KVM and dumping that data into Elasticsearch. This is essential since we are going to use all that data to create visualizations and dashboards.

Creating charts in Kibana

Now, open Kibana in a browser using **localhost:5601** as the address. There should be an icon-based menu on the left-hand side of the screen. Go to **Stack management** and take a look at **Elasticsearch index management**.

There should be an active index named **metricbeat-*<somenumber>***. In this particular example, *<somenumber>* will be the current version of metricbeat and the date of the first entry in the log file. This is completely arbitrary and is just a default that ensures you know when this instance was started.

In the same line as this name, there should be some numbers: what we are interested in is the docs count – the number of objects that database holds. For the time being, if it's not zero, we are okay.

Now, go to the **Dashboard** page and open the **Metricbeat System Overview ECS** dashboard. It will show a lot of visual widgets representing CPU, memory, disk, and network usage:

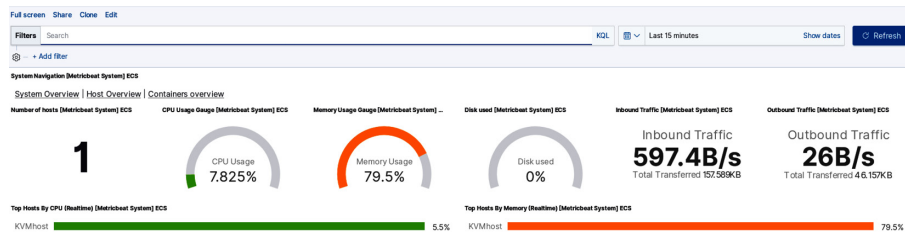


Figure 14.7 – Overview of the ECS dashboard

Now, you can click on **Host Overview** and view even more data about your system. Try playing with the dashboard and different settings. One of the most interesting items on this dashboard is the one in the upper-right part of the screen – the one that defines the timespan that we are interested in. We can either create our own or use one of the presets, such as **last 15 minutes**. After you click the **Refresh** button, new data should show on the page.

With that, you now know enough about Kibana to get started, but we still are unable to visualize KVM data. The next step is to create a dashboard that will cover that.

But before we do that, think about what you can do with only what we've learned so far. Not only can you monitor the local system that has your KVM stack installed, but you can also monitor any system that is able to run Metricbeat. The only thing that you need to know is the IP address of the ELK stack, so that you can send data to it. Kibana will automatically deal with visual-

izing all the different data from different systems, as we will see later.

Creating custom utilization reports

Since version 7, Elastic stack has introduced mandatory checks that are designed to ensure minimum security and functionality compliance, especially once we start using ELK in production.

At first glance, these checks may confuse you – the installation that we guided you through will work, and suddenly, as you try to configure some settings, everything will fail. This is intentional.

In previous versions, these checks were performed but were flagged as warnings if a configuration item was missed or misconfigured.

Starting from version 7, these checks will trigger an error when the system is in production and not configured correctly. This state automatically means that your installation will not work if it's not configured properly.

ELK has two distinct modes of operation: *development* and *production*. On the first installation, it is assumed that you are in development mode, so most of the functionality simply works out of the box.

Things change a lot once you go into production mode – security settings and other configuration options need to be explicitly set in order for the stack to function.

The trick is that there is no explicit mode change – production settings and checks associated with them are triggered by some settings in the configuration. The idea is that once you reconfigure something that can be important from a security standpoint, you need to reconfigure everything correctly. This will prevent you from forgetting something that can be a big problem in production and force you to have at least a stable configuration to start from. There is a switch to disable checks, but it is not recommended in any circumstances.

The main thing to pay attention to is the binding interface – the default installation binds everything to **localhost** or a local loopback interface, which is completely fine for production. Once your Elasticsearch is capable of forming a cluster and it can be triggered by simply reconfiguring the network address for HTTP and transport communication, you have to pay attention to the checks and reconfigure the whole system in order to make it work. Please consult the documentation available on <https://www.elastic.co/> for

more information, starting with

<https://www.elastic.co/guide/index.html>.

For example, configuring clusters in the Elastic stack and all that it entails is way out of the scope of this book – we are going to stay within the realm of a *single-node cluster* in our configuration. This solution was specifically created for situations that can work with a single node or, more precisely, a single machine instance that covers all the functionality of a stack. In a normal deployment, you will run Elastic stack in a cluster, but implementation details will be something determined by your configuration and its needs.

We need to warn you of two crucial points – firewall and SELinux settings are up to you. All the services use standard TCP to communicate. Don't forget that for the services to run, the network has to be configured correctly.

Now that we've gotten that out of the way, let's answer one simple question: what do we need to do to make the Elastic stack work with more than one server? Let's discuss this scenario, bit by bit.

Elasticsearch

Go to the configuration file
(`/etc/elasticsearch/elasticsearch.yml`) and
add a line in the discovery section:

```
discovery.type: single-node
```

Using this section is not mandatory, but it helps
when you must go back to the configuration
later.

This option will tell Elasticsearch that you will
have only one node in the cluster, and it will
make Elasticsearch ignore all the checks associ-
ated with the cluster and its network. This set-
ting will also make this node the master node au-
tomatically since Elasticsearch depends on hav-
ing master nodes that control everything in the
cluster.

Change the setting under **network.host**: so that
it points to the IP address of the interface
Elasticsearch is going to be available on. By de-
fault, it points to localhost and is not visible from
the network.

Restart the Elasticsearch service and make sure
it is running and not generating errors:

```
sudo systemctl restart  
elasticsearch.service
```

Once you have it working, check whether the
service is behaving normally from the local ma-

chine. The easiest way is to do this is as follows:

```
curl -XGET <ip_address>:9200
```

The response should be **.json** formatted text containing information about the server.

Important note

The Elastic stack has three (or four) parts or services. In all our examples, three of them (Logstash, Elasticsearch, and Kibana) were running on the same server, so no additional configuration was necessary to accommodate network communication. In a normal configuration, these services would probably run on independent servers and in multiple instances, depending on the workload and configuration of the service we are trying to monitor.

Logstash

The default installation for Logstash is a file named **logstash-sample.conf** in the **/etc/logstash** folder. This contains a simple Logstash pipeline to be used when we are using Logstash as the primary destination for beats. We will come to this later, but for the time being, copy this file to **/etc/logstash/conf.d/logstash.conf** and change the address of the Elasticsearch server in

the file you just copied. It should look something like this:

```
hosts => ["http://localhost:9200"].
```

Change **localhost** to the correct IP address of your server. This will make Logstash listen on port **5044** and forward the data to Elasticsearch. Restart the service and verify that it runs:

```
sudo systemctl restart  
logstash.service
```

Now, let's learn how to configure Kibana.

Kibana

Kibana also has some settings that need to be changed, but when doing so, there are a couple of things to remember about this service:

- By itself, Kibana is a service that serves visualizations and data over the HTTP protocol (or HTTPS, depending on the configuration).
- At the same time, Kibana uses Elasticsearch as its backend in order to get and work with data. This means that there are two IP addresses that we must care about:
 - a) The first one is the address that will be used to show Kibana pages. By default, this is localhost on port **5601**.
 - b) The other IP address is the Elasticsearch service that will deal with the queries. The default

for this is also localhost, but it needs to be changed to the IP address of the Elasticsearch server.

The file that contains configuration details is `/etc/kibana/kibana.yml` and you need to at least make the following changes:

- **server.host:** This needs to point to the IP address where Kibana is going to have its pages.
- **elasticsearch.hosts:** This needs to point to the host (or a cluster, or multiple hosts) that are going to perform queries.

Restart the service, and that's it. Now, log into Kibana and test whether everything works.

To get you even more familiarized with Kibana, we will try and establish some basic system monitoring and show how we can monitor multiple hosts. We are going to configure two *beats*: Metricbeat and Filebeat.

We already configured Metricbeat, but it was for localhost, so let's fix that first. In the `/etc/metricbeat/metricbeat.yml` file, reconfigure the output in order to send data to the **elasticsearch** address. You only need to change the host IP address since everything else stays the same:

```
# Array of hosts to connect to  
Hosts: ["Your-host-IP-address:9200"]
```

Make sure that you change **Your-host-IP-address** to the IP address you're using.

Configuring filebeat is mostly the same; we need to use `/etc/filebeat/filebeat.yml` to configure it. Since all the beats use the same concepts, both filebeat and metricbeat (as well as other beats) use modules to provide functionality. In both, the core module is named **system**, so enable it using the following command in filebeat:

```
filebeat modules enable system
```

Use the following command for metricbeat:

```
metricbeat modules enable system
```

We mentioned this previously, in the first example, but you can test your configuration by running the following command:

```
filebeat test config
```

You can also use the following command:

```
metricbeat test config
```

Both beats should say that the config is **ok**.

Also, you can check the output settings, which will show you what the output settings actually are and how they work. If you are configuring the system using only this book, you should have

a warning come up to remind you there is no TLS protection for the connection, but otherwise, the outputs should work on the IP address that you set in the configuration file.

To test the outputs, use the following command:

```
filebeat test output
```

You can also use the following command:

```
metricbeat test output
```

Repeat all this for every system that you intend to monitor. In our example, we have two systems: one that is running KVM and another that is running Kibana. We also have Kibana set up on the other system to test syslog and the way it notifies us of the problems it notices.

We need to configure filebeat and metricbeat to send data to Kibana. We'll edit the **filebeat.yml** and **metricbeat.yml** files for that purpose, by changing the following portion of both files:

```
setup.kibana
  host: "Your-Kibana-Host-IP:5601"
```

Before running beats, on a fresh installation, you need to upload dashboards to Kibana. You only need to do this once for each Kibana installation, and you only need to do this from one of the systems you are monitoring – templates will work, regardless of the system they were uploaded

from; they will just deal with data that is coming into Elasticsearch.

To do this, use the following command:

```
filebeat setup
```

You also need to use the following command:

```
metricbeat setup
```

This will take a couple of seconds or even a minute, depending on your server and client. Once it says that it created the dashboards, it will display all the dashboards and settings it created.

Now, you are almost ready to go through all the data that Kibana will display:

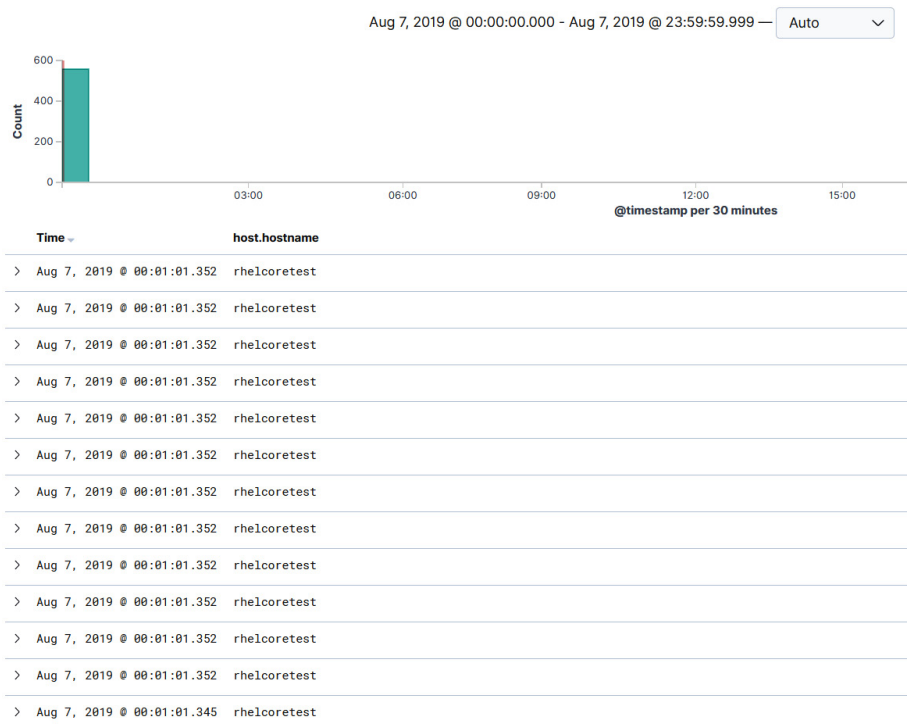


Figure 14.8 – Excerpt from the Kibana dashboard

Before we start, there's something else you need to know about time and timestamps. The date/time picker in the top-right corner will let you choose either your own timespan or one of the predefined intervals:

The screenshot shows the Kibana date/time picker interface. At the top, there is a 'Quick select' section with a dropdown menu set to 'Last', a text input field containing '15', a dropdown menu set to 'minutes', and an 'Apply' button. Below this is a 'Commonly used' section with a grid of links: 'Today', 'This week', 'Last 15 minutes', 'Last 30 minutes', 'Last 1 hour', 'Last 24 hours', 'Last 7 days', 'Last 30 days', 'Last 90 days', and 'Last 1 year'. Underneath is a 'Recently used date ranges' section with links for 'Last 24 hours', 'Today', 'Aug 6, 2019 @ 03:47:17.882 to Aug 6, 2019 @ 22:37:06.182', 'Last 15 minutes', and 'Last 1 hour'. At the bottom is a 'Refresh every' section with a text input field set to '0', a dropdown menu set to 'seconds', and a 'Start' button.

Figure 14.9 – Date/time picker

Important note

Always remember that the time that's shown is local to the browser's/machine's time zone you are accessing Kibana from.

All the timestamps in the logs are *local* to the machine that is sending the logs. Kibana will try and match time zones and translate the resulting timestamps, but if there is a mismatch in the actual time settings on the machines you are monitoring, there is going to be a problem trying to establish a timeline of events.

Let's presume you got filebeat and metricbeat running. What can you do with these? As it turns out, a lot:

- The first thing is discovering what is in your data. Press the **Discover** button in Kibana (it looks like a small compass). Some data should show on the right if everything is okay.
- To the right of the icon you just clicked on, a vertical space will fill up with all the attributes that Kibana got from the data. If you do not see anything or something is missing, remember that the time span you select narrows down the data that will get shown in this view. Try readjusting the interval to **Last 24 hours** or **Last 30 days**.

Once the list of attributes shows up, you can quickly establish how many times each shows up in the data you just selected – just click on any attribute and select **Visualize**. Also note that once you click on the attribute, Kibana shows you the top five distinct values in the last 500 records. This is a very useful tool if you need to know, for example, which hosts are showing data, or how many different OS versions there are.

The visualization of particular attributes is just a start – notice how, once you hover over an attribute name, a button called **Add** appears? Try

clicking it. A table will start forming on the right, filled with just the attributes you selected, sorted by timestamp. By default, these values are not auto-refreshed, so the timestamps will be fixed. You can choose as many attributes as you want and save this list or open it later.

The next thing we need to look at is individual visualizations. We are not going to go into too many details, but you can create your own visualizations out of the datasets using predefined visualization types. At the same time, you are not limited to using only predefined things – using JSON and scripting is also possible, for even more customization.

The next thing we need to learn about is dashboards.

Depending on a particular dataset, or to be more precise, on the particular set of machines you are monitoring, some of them will have attributes that cover things only a particular machine does or has. One example is virtual machines on AWS – they will have some information that is useful only in the context of AWS. This is not important in our configuration, but you need to understand that there may be some attributes in the data that are unique for a particular set of machines. For starters, choose one

of the system metrics; either **System Navigation** ECS for metricbeat or **Dashboards** ECS for filebeat.

These dashboards show a lot of information about your systems in a lot of ways. Try clicking around and see what you can deduce.

The metricbeat dashboard is more oriented toward running systems and keeping an eye on memory and CPU allocation. You can click and filter a lot of information, and have it presented in different ways. The following is a screenshot of metricbeat so that you can get a rough idea of what it looks like:

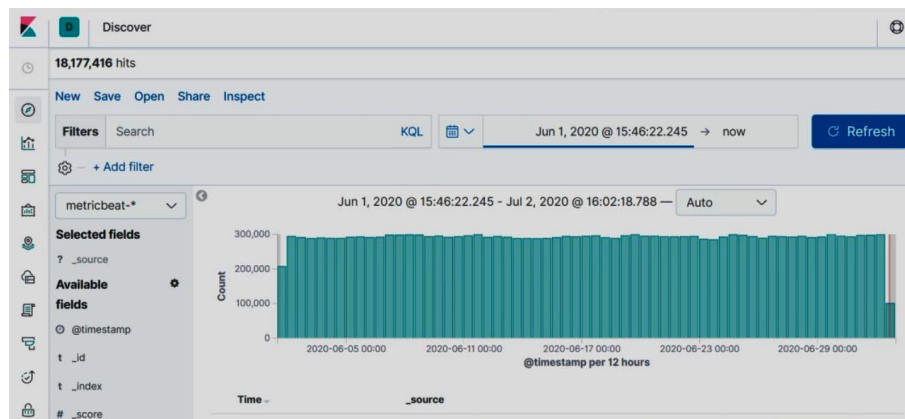


Figure 14.10 – metricbeat dashboard

The filebeat dashboard is more oriented toward analyzing what happened and establishing trends. Let's check a couple of excerpts from the filebeat dashboard, starting with the syslog entries part:

Syslog logs [Filebeat System] ECS		
Time	host.hostname	process.name
> Aug 6, 2019 @ 20:35:01.000	localhost	kibana
> Aug 6, 2019 @ 20:35:01.000	localhost	auditbeat
> Aug 6, 2019 @ 20:35:01.000	rhelcoretest	logstash
> Aug 6, 2019 @ 20:34:59.000	localhost	dbus-daemon
> Aug 6, 2019 @ 20:34:59.000	localhost	systemd
> Aug 6, 2019 @ 20:34:59.000	localhost	journal
> Aug 6, 2019 @ 20:34:59.000	localhost	dbus-daemon

Figure 14.11 – filebeat syslog entries part

At first glance, you can notice a couple of things. We are showing data for two systems, and the data is partial since it covers a part of the interval that we set. Also, we can see that some of the processes are running and generating logs more frequently than others. Even if we do not know anything about the particular system, we can now see there are some processes that show up in logs, and they probably shouldn't:

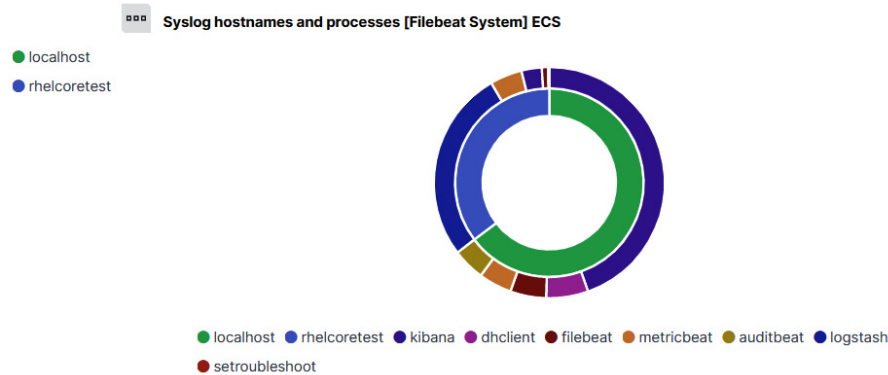


Figure 14.12 – filebeat interactive doughnut chart

Let's take a look at **setroubleshoot**. Click on the process name. In the window that opens, click on the magnifying glass. This isolates only this

process and shows only its logs at the bottom of the screen.

We can quickly see on which host – including how often and why – **setroubleshoot** is writing logs to. This is a quick way to spot potential problems. In this particular case, some action should obviously be taken on this system to reconfigure SELinux since it generates exceptions and stops some applications from accessing files.

Let's move along the vertical navigation bar and point out some other interesting functionalities.

Going from top to bottom, the next big functionality is **Canvas** – it enables us to create live presentations using data from the dataset we are collecting. The interface is similar to what can be expected from other presentation programs, but the accent is on using data directly in slides and generating slides in almost real time.

The next is **Maps**. This is a new addition to version 7.0 and allows us to create a geographic presentation of data.

Machine learning is next – it enables you to manipulate data and use it to "train" filters and create pipelines out of them.

Infrastructure is also interesting – when we mentioned dashboards, we were talking about flexibility and customization. Infrastructure is a module that enables us to do real-time monitoring with minimal effort and observe important metrics. You can see important data either as a table, a balloon-like interface, or as a graph. Data can be averaged or presented in other ways, and all that is done through a highly intuitive interface.

Heartbeat is another of these highly specialized boards – as its name suggests, it is the easiest way to track and report on uptime data, and to quickly notice if something has gone offline. Inventory hosts require the Heartbeat service to be installed on each system we intend to monitor.

SIEM deserves a more thorough explanation: if we think about dashboards as being multipurpose, SIEM is the exact opposite; it is created to be able to track all the events on all the systems that can be categorized as security-related. This module will parse the data when searching for IPs, network events, sources, destinations, network flows, and all other data, and create simple to understand reports regarding what is happening on the machines you are monitoring. It even

offers anomaly detection, a feature that enables the Elastic stack to serve as a solution for advanced security purposes. This feature is a paid one and requires the highest-paid tier to function.

Stack monitor is another notable board as it enables you to actually see what is happening in all the different parts of the Elastic stack. It will show the status of all the services, their resource allocation, and license status. The **Logs** feature is especially useful since it tracks how many logs of what type the stack is generating, and it can quickly point to problems if there are any.

This module also generates statistics for services, enabling us to understand how the system can be optimized.

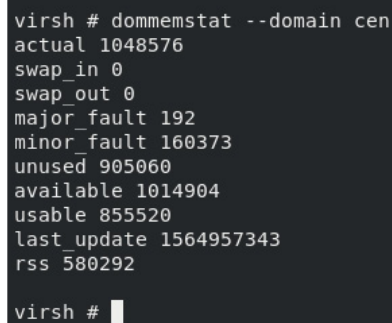
Management, the last icon at the bottom, was already mentioned – it enables the management of the cluster and its parts. This is the place where we can see whether there are any indices we are expecting, whether the data flowing in, whether can we optimize something, and so on. This is also the place we can manage licenses and create snapshots of system configuration.

ELK and KVM

Last but not least, let's create a system gauge that is going to show us a parameter from a KVM hypervisor, and then visualize it in a couple of ways. The prerequisites for this are a running KVM hypervisor, metricbeat with the KVM module installed, and an Elastic stack configuration that supports receiving data from metricbeat. Let's go through ELK's configuration for this specific use case:

1. First, go to the hypervisor and open a **virsh** shell. List all the available domains, choose a domain, and use the **dommemstat --domain <domain_name>** command.

The result should be something like this:



```
virsh # dommemstat --domain cen
actual 1048576
swap_in 0
swap_out 0
major_fault 192
minor_fault 160373
unused 905060
available 1014904
usable 855520
last_update 1564957343
rss 580292
virsh #
```

Figure 14.13 – dommemtest for a domain

2. Open Kibana and log in, go to the **Discover** tab, and select **metric*** as the index we are working with. The left column should populate with attributes that are in the dataset that metricbeat sends to this Kibana instance. Now, look at the attributes and select a couple of them:

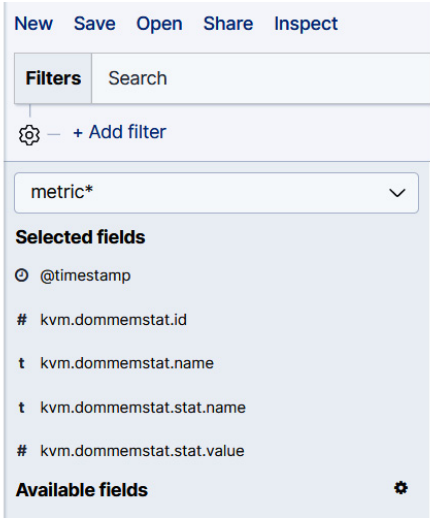


Figure 14.14 – Selecting attributes in Kibana

You can select fields using a button that will show up as soon as you hover the mouse cursor over any field. The same goes for deselecting them:

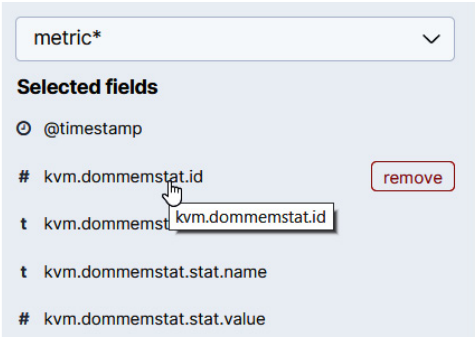


Figure 14.15 – Adding attributes in Kibana

3. For now, let's stick with the ones we selected.

To the right of the column, a table is formed that contains only the fields you selected, enabling you to check the data that the system is receiving. You may need to scroll down to see the actual information since this table will display all the data that was received that has at least one item that has a value. Since one of the fields is always a timestamp, there will be a lot

of rows that will not contain any useful data
for our analysis:

Aug 8, 2019 @ 23:22:49.313	Aug 8, 2019 @ 23:22:49.313	2	cen	actualballoon	1,048,576
Aug 8, 2019 @ 23:22:49.313	Aug 8, 2019 @ 23:22:49.313	2	cen	swpin	0
Aug 8, 2019 @ 23:22:49.313	Aug 8, 2019 @ 23:22:49.313	2	cen	swapout	0
Aug 8, 2019 @ 23:22:49.313	Aug 8, 2019 @ 23:22:49.313	2	cen	majorfault	192
Aug 8, 2019 @ 23:22:49.313	Aug 8, 2019 @ 23:22:49.313	2	cen	minorfault	160,373
Aug 8, 2019 @ 23:22:49.313	Aug 8, 2019 @ 23:22:49.313	2	cen	unused	905,060
Aug 8, 2019 @ 23:22:49.313	Aug 8, 2019 @ 23:22:49.313	2	cen	available	1,014,904
Aug 8, 2019 @ 23:22:49.313	Aug 8, 2019 @ 23:22:49.313	2	cen	usable	855,520
Aug 8, 2019 @ 23:22:49.313	Aug 8, 2019 @ 23:22:49.313	2	cen	lastupdate	1,564,957,343
Aug 8, 2019 @ 23:22:49.313	Aug 8, 2019 @ 23:22:49.313	2	cen	rss	500,292

Figure 14.16 – Checking the selected fields

What we can see here is that we get the same data that was the result of running the command line on the monitored server.

What we need is a way to use these results as data to show our graphs. Click on the **Save** button at the top left of the screen. Use a name that you will be able to find later; we used **dom-memstat**. Save the search.

4. Now, let's build a gauge that will show us the real-time data and a quick visualization of one of the values. Go to **Visualize** and click **Create new visualization**:

Visualizations Create new visualization

Q Search...

<input type="checkbox"/> Title	Type	Actions
<input type="checkbox"/> Bytes Timeline [Filebeat NATS] ECS	Line	
<input type="checkbox"/> Cache Hits, Misses [Metricbeat CoreDNS] ECS	Line	
<input type="checkbox"/> ASA Events Over Time [Filebeat Cisco]	Vertical Bar	
<input type="checkbox"/> ASA Firewall Blocked by Source [Filebeat Cisco]	Data Table	
<input type="checkbox"/> ASA Flows by Network Bytes [Filebeat Cisco]	Vertical Bar	
<input type="checkbox"/> ASA Top ACL by Blocked [Filebeat Cisco]	Data Table	
<input type="checkbox"/> AWS Cloudwatch ECS CPU Available	TSVB	
<input type="checkbox"/> AWS Cloudwatch ECS Memory Available	TSVB	
<input type="checkbox"/> AWS Cloudwatch ELB Latency	TSVB	
<input type="checkbox"/> AWS Cloudwatch ELB Request Count Top5	TSVB	

Rows per page: 10 < 1 2 3 4 5 ... 49 >

Figure 14.17 – Creating a new visualization

Choose the **area** graph. Then, on the next screen, find and select our source for the data:

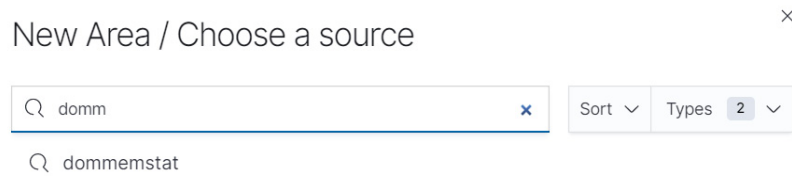


Figure 14.18 – Selecting a visualization source

This is going to create a window that has all the settings on the left and the end result on the right. At the moment, what we can see makes no sense, so let's configure what we need in order to show our data. There are a couple of ways to accomplish what we want: we are going to use a histogram and filters to quickly display how our value for unused memory changed over time.

5. We are going to configure the y axis to show average data for **kvm.dommemstat.stat.value**, which is the attribute that holds our data. Select **Average** under **Aggregation** and **kvm.dommemstat.stat.value** as the field we are aggregating. You can create a custom label if you want to:

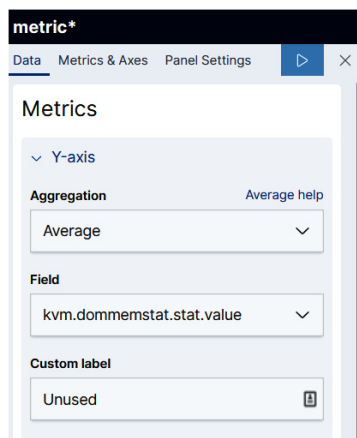


Figure 18.19 – Selecting metric properties

This is still not right; we need to add a time-stamp to see how our data changed over time.

We need to add a **Date Histogram** type to the x axis and use it:

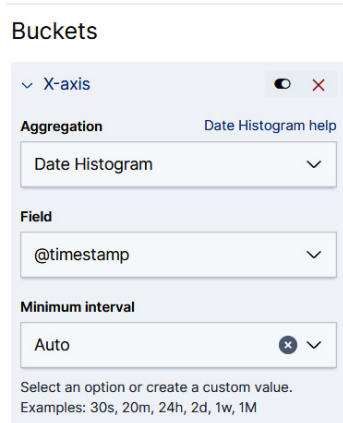


Figure 14.20 – Choosing an aggregation type

6. Before we finish this visualization, we need to add a filter. The problem with data that is received from the KVM metricbeat module is that it uses one attribute to hold different data – if we want to know what the number in the file we are displaying actually means, we need to read its name from

kvm.dommemstat.stat.name. To accomplish this, just create a filter called **kvm.dommemstat.stat.name:"unused"**.

After we refresh the visualization, our data should be correctly visualized on the right:

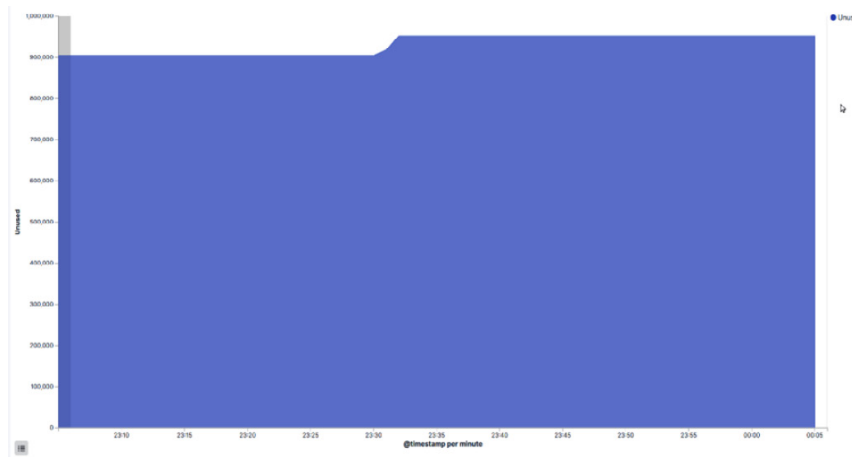


Figure 14.21 – Correct visualization

7. We need to save this visualization using the **Save** button, give it a name that we will be able to find later, and repeat this process but instead of filtering for **unused**, filter for **usable**. Leave all the settings identical to the first visualization.

Let's build a dashboard. Open the **Dashboard** tab and click on **Add new dashboard** on the first screen. Now, add our two visualizations to this dashboard. You just need to find the right visualization and click on it; it will show up on the dashboard.

As a result, we have a couple of simple dashboards running:

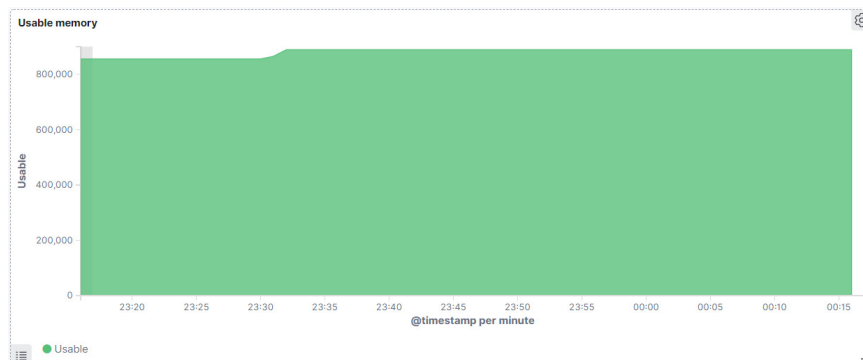


Figure 14.22 – Finished dashboard showing usable memory

The second dashboard – in the UI, which is actually right next to the first one – is the unused memory dashboard:

Figure 14.23 – Finished dashboard showing unused memory

8. Save this dashboard so that you can use it later. All the elements of the dashboard can be customized, and the dashboard can consist of any number of visualizations. Kibana lets you customize almost everything you see and combine a lot of data on one screen for easy monitoring. There is only one thing we need to change to make this a good monitoring dashboard, and that is to make it autorefresh. Click on the calendar icon on the right-hand side of the screen and select the auto refresh interval. We decided on **5 seconds**:

Figure 14.24 – Selecting time-related parameters

Now that we've done this, we can reflect on the fact that building this dashboard was really straightforward and easy. This only took us a couple of minutes, and it's easy to read. Imagine going through hundreds of megabytes of log files in text mode compared to this. There's really no contest, as we were able to use our previously deployed ELK stack to monitor information about KVM, which was the whole point of this chapter.

Summary

What Kibana enables you to do is create custom dashboards that can show you data for different machines side by side, so KVM is just one of the many options we have. Depending on your needs, you can display, for example, disk usage for a KVM hypervisor and all the hosts running on it, or some other metric. The Elastic stack is a flexible tool, but as with all things, it requires time to master. This chapter only covered the bare basics of Elastic configuration, so we strongly recommend further reading on this topic – alongside KVM, ELK can be used to monitor almost everything that produces any kind of data.

The next chapter is all about performance tuning and optimization for KVM virtual machines, a subject that we didn't really touch upon. There's quite a lot to be discussed – virtual machine compute sizes, optimizing performance, disks, storage access and multipathing, optimizing kernels, and virtual machine settings, just to name a few. All these subjects will be more important the larger our environment becomes.

Questions

1. What do we use metricbeat for?
2. Why do we use Kibana?
3. What is the basic prerequisite before installing the ELK stack?
4. How do we add data to Kibana?

Further reading

Please refer to the following links for more information regarding what was covered in this chapter:

- ELK stack: <https://www.elastic.co/what-is/elk-stack>
- ELK stack documentation: <https://www.elastic.co/guide/index.html>
- Kibana documentation: <https://www.elastic.co/guide/en/kibana/current/index.html>
- Metricbeat documentation: <https://www.elastic.co/guide/en/beats/metricbeat/current/index.html>