Elasticsearch, Logstash, Kibana (ELK) Docker image documentation

This web page documents how to use the sebp/elk Docker image, which provides a convenient centralised log server and log management web interface, by packaging Elasticsearch, Logstash, and Kibana, collectively known as ELK.

Contents

Prerequisites

To run a container using this image, you will need the following:

- **Docker**

  Install Docker, either using a native package (Linux) or wrapped in a virtual machine (Windows, OS X – e.g. using Boot2Docker or Vagrant).
- **A minimum of 4GB RAM assigned to Docker**

  Elasticsearch alone needs at least 2GB of RAM to run.

  With Docker for Mac, the amount of RAM dedicated to Docker can be set using the UI: see How to increase docker-machine memory Mac (Stack Overflow).
- **A limit on mmap counts equal to 262,144 or more**

  !! This is the most frequent reason for Elasticsearch failing to start since Elasticsearch version 5 was released.

  On Linux, use `sysctl vm.max_map_count` on the host to view the current value, and see Elasticsearch's documentation on virtual memory for guidance on how to change this value. Note that the limits **must be changed on the host**; they cannot be changed from within a container.

  If using Docker for Mac, then you will need to start the container with the `MAX_MAP_COUNT` environment variable (see Overriding start-up variables) set to at least 262144 (using e.g. `docker`'s `-e` option) to make Elasticsearch set the limits on mmap counts at start-up time.
- **Access to TCP port 5044 from log-emitting clients**

  Other ports may need to be explicitly opened: see Usage for the complete list of ports that are exposed.

Installation

To pull this image from the Docker registry, open a shell prompt and enter:

```
$ sudo docker pull sebp/elk
```

Note – This image has been built automatically from the source files in the source Git repository on GitHub. If you want to build the image yourself, see the Building the image section.

Pulling specific version combinations

Specific version combinations of Elasticsearch, Logstash and Kibana can be pulled by using tags.

For instance, the image containing Elasticsearch 1.7.3, Logstash 1.5.5, and Kibana 4.1.2 (which is the last image using the Elasticsearch 1.x and Logstash 1.x branches) bears the tag `E1L1K4`, and can therefore be pulled using `sudo docker pull sebp/elk:E1L1K4`.

The available tags are listed on Docker Hub's sebp/elk image page or GitHub repository page.

By default, if no tag is indicated (or if using the tag `latest`), the latest version of the image will be pulled.

Usage

Run a container from the image with the following command:

```
$ sudo docker run -p 5601:5601 -p 9200:9200 -p 5044:5044 -it --name elk sebp/elk
```

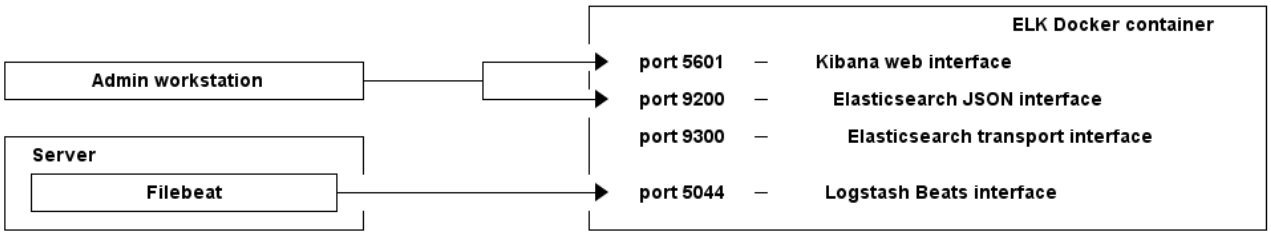
Note – The whole ELK stack will be started. See the *Starting services selectively* section to selectively start part of the stack.

This command publishes the following ports, which are needed for proper operation of the ELK stack:

- 5601 (Kibana web interface).
- 9200 (Elasticsearch JSON interface).
- 5044 (Logstash Beats interface, receives logs from Beats such as Filebeat – see the *Forwarding logs with Filebeat* section).

Note – The image also exposes Elasticsearch's transport interface on port 9300. Use the `-p 9300:9300` option with the `docker` command above to publish it. This transport interface is notably used by Elasticsearch's Java client API, and to run Elasticsearch in a cluster.

The figure below shows how the pieces fit together.



Access Kibana's web interface by browsing to `http://<your-host>:5601`, where `<your-host>` is the hostname or IP address of the host Docker is running on (see note), e.g. `localhost` if running a local native version of Docker, or the IP address of the virtual machine if running a VM-hosted version of Docker (see note).

Note – To configure and/or find out the IP address of a VM-hosted Docker installation, see https://docs.docker.com/installation/windows/ (Windows) and https://docs.docker.com/installation/mac/ (OS X) for guidance if using Boot2Docker. If you're using Vagrant, you'll need to set up port forwarding (see https://docs.vagrantup.com/v2/networking/forwarded_ports.html.

You can stop the container with `^C`, and start it again with `sudo docker start elk`.

As from Kibana version 4.0.0, you won't be able to see anything (not even an empty dashboard) until something has been logged (see the *Creating a dummy log entry* sub-section below on how to test your set-up, and the *Forwarding logs* section on how to forward logs from regular applications).

When filling in the index pattern in Kibana (default is `logstash-*`), note that in this image, Logstash uses an output plugin that is configured to work with Beat-originating input (e.g. as produced by Filebeat, see Forwarding logs with Filebeat) and that logs will be indexed with a `<beatname>-` prefix (e.g. `filebeat-` when using Filebeat).

Running the container using Docker Compose

If you're using Docker Compose to manage your Docker services (and if not you really should as it will make your life much easier!), then you can create an entry for the ELK Docker image by adding the following lines to your `docker-compose.yml` file:

```
elk:
  image: sebp/elk
  ports:
    - "5601:5601"
    - "9200:9200"
    - "5044:5044"
```

You can then start the ELK container like this:

```
$ sudo docker-compose up elk
```

Running the container using Kitematic

Windows and OS X users may prefer to use a simple graphical user interface to run the container, as provided by Kitematic, which is included in the Docker Toolbox.

After starting Kitematic and creating a new container from the sebp/elk image, click on the *Settings* tab, and then on the *Ports* sub-tab to see the list of the ports exposed by the container (under *DOCKER PORT*) and the list of IP addresses and ports they are published on and accessible from on your machine (under *MAC IP:PORT*).

You may for instance see that Kibana's web interface (which is exposed as port 5601 by the container) is published at an address like 192.168.99.100:32770, which you can now go to in your browser.

Note – The rest of this document assumes that the exposed and published ports share the same number (e.g. will use `http://<your-host>:5601/` to refer to Kibana's web interface), so when using Kitematic you need to make sure that you replace both the hostname with the IP address *and* the exposed port with the published port listed by Kitematic (e.g. `http://192.168.99.100:32770` in the previous example).

Creating a dummy log entry

If you haven't got any logs yet and want to manually create a dummy log entry for test purposes (for instance to see the dashboard), first start the container as usual (`sudo docker run ...` or `docker-compose up ...`).

In another terminal window, find out the name of the container running ELK, which is displayed in the last column of the output of the `sudo docker ps` command.

```
$ sudo docker ps
CONTAINER ID        IMAGE                  ...   NAMES
86aea21cab85        elkdocker_elk:latest   ...   elkdocker_elk_1
```

Open a shell prompt in the container and type (replacing `<container-name>` with the name of the container, e.g. `elkdocker_elk_1` in the example above):

```
$ sudo docker exec -it <container-name> /bin/bash
```

At the prompt, enter:

```
# /opt/logstash/bin/logstash --path.data /tmp/logstash/data \
    -e 'input { stdin { } } output { elasticsearch { hosts => ["localhost"] } }'
```

Wait for Logstash to start (as indicated by the message `The stdin plugin is now waiting for input:`), then type some dummy text followed by Enter to create a log entry:

```
this is a dummy entry
```

Note – You can create as many entries as you want. Use `^C` to go back to the bash prompt.

If you browse to `http://<your-host>:9200/_search?pretty` (e.g. http://localhost:9200/_search?pretty for a local native instance of Docker) you'll see that Elasticsearch has indexed the entry:

```
{
  ...
  "hits": {
    ...
    "hits": [ {
      "_index": "logstash-...",
      "_type": "logs",
      ...
      "_source": { "message": "this is a dummy entry", "@version": "1", "@timestamp": ... }
    } ]
  }
}
```

You can now browse to Kibana's web interface at `http://<your-host>:5601` (e.g. http://localhost:5601 for a local native instance of Docker).

Make sure that the drop-down "Time Filter field name" field is pre-populated with the value `@timestamp`, then click on "Create", and you're good to go.

Starting services selectively

By default, when starting a container, all three of the ELK services (Elasticsearch, Logstash, Kibana) are started.

The following environment variables may be used to selectively start a subset of the services:

- `ELASTICSEARCH_START`: if set and set to anything other than `1`, then Elasticsearch will not be started.
- `LOGSTASH_START`: if set and set to anything other than `1`, then Logstash will not be started.
- `KIBANA_START`: if set and set to anything other than `1`, then Kibana will not be started.

For example, the following command starts Elasticsearch only:

```
$ sudo docker run -p 5601:5601 -p 9200:9200 -p 5044:5044 -it \
    -e LOGSTASH_START=0 -e KIBANA_START=0 --name elk sebp/elk
```

Note that if the container is to be started with Elasticsearch *disabled*, then:

- If Logstash is enabled, then you need to make sure that the configuration file for Logstash's Elasticsearch output plugin (`/etc/logstash/conf.d/30-output.conf`) points to a host belonging to the Elasticsearch cluster rather than `localhost` (which is the default in the ELK image, since by default Elasticsearch and Logstash run together), e.g.:

```
output {
  elasticsearch { hosts => ["elk-master.example.com"] }
}
```

- Similarly, if Kibana is enabled, then Kibana's `kibana.yml` configuration file must first be updated to make the `elasticsearch.url` setting (default value: `"http://localhost:9200"`) point to a running instance of Elasticsearch.

Overriding start-up variables

The following environment variables can be used to override the defaults used to start up the services:

- `TZ`: the container's time zone (see list of valid time zones), e.g. `America/Los_Angeles` (default is `Etc/UTC`, i.e. UTC).
- `ES_HEAP_SIZE`: Elasticsearch heap size (default is 256MB min, 1G max)

  Specifying a heap size – e.g. `2g` – will set both the min and max to the provided value. To set the min and max values separately, see the `ES_JAVA_OPTS` below.
- `ES_JAVA_OPTS`: additional Java options for Elasticsearch (default: `""`)

  For instance, to set the min and max heap size to 512MB and 2G, set this environment variable to `-Xms512m -Xmx2g`.
- `ES_CONNECT_RETRY`: number of seconds to wait for Elasticsearch to be up before starting Logstash and/or Kibana (default: `30`)
- `ES_PROTOCOL`: protocol to use to ping Elasticsearch's JSON interface URL (default: `http`)

  Note that this variable is only used to test if Elasticsearch is up when starting up the services. It is not used to update Elasticsearch's URL in Logstash's and Kibana's configuration files.
- `CLUSTER_NAME`: the name of the Elasticsearch cluster (default: automatically resolved when the container starts if Elasticsearch requires no user authentication).

  The name of the Elasticsearch cluster is used to set the name of the Elasticsearch log file that the container displays when running. By default the name of the cluster is resolved automatically at start-up time (and populates `CLUSTER_NAME`) by querying Elasticsearch's REST API anonymously. However, when Elasticsearch requires user authentication (as is the case by default when running X-Pack for instance), this query fails and the container stops as it assumes that Elasticsearch is not running properly. Therefore, the `CLUSTER_NAME` environment variable can be used to specify the name of the cluster and bypass the (failing) automatic resolution.
- `LS_HEAP_SIZE`: Logstash heap size (default: `"500m"`)
- `LS_OPTS`: Logstash options (default: `"--auto-reload"` in images with tags `es231_l231_k450` and `es232_l232_k450`, `""` in `latest`; see Breaking changes)
- `NODE_OPTIONS`: Node options for Kibana (default: `"--max-old-space-size=250"`)
- `MAX_MAP_COUNT`: limit on mmap counts (default: system default)

Warning – This setting is system-dependent: not all systems allow this limit to be set from within the container, you may need to set this from the host before starting the container (see Prerequisites).

- `MAX_OPEN_FILES`: maximum number of open files (default: system default; Elasticsearch needs this amount to be equal to at least 65536)
- `KIBANA_CONNECT_RETRY`: number of seconds to wait for Kibana to be up before running the post-hook script (see Pre-hooks and post-hooks) (default: `30`)

As an illustration, the following command starts the stack, running Elasticsarch with a 2GB heap size and Logstash with a 1GB heap size:

```
$ sudo docker run -p 5601:5601 -p 9200:9200 -p 5044:5044 -it \
    -e ES_HEAP_SIZE="2g" -e LS_HEAP_SIZE="1g" --name elk sebp/elk
```

Pre-hooks and post-hooks

Before starting the ELK services, the container will run the script at `/usr/local/bin/elk-pre-hooks.sh` if it exists and is executable.

This can in particular be used to expose custom environment variables (in addition to the default ones supported by the image) to Elasticsearch and Logstash by amending their corresponding `/etc/default` files.

For instance, to expose the custom `MY_CUSTOM_VAR` environment variable to Elasticsearch, add an executable `/usr/local/bin/elk-pre-hooks.sh` to the container (e.g. by `ADD`-ing it to a custom `Dockerfile` that extends the base image, or by bind-mounting the file at runtime), with the following contents:

```
cat << EOF >> /etc/default/elasticsearch
MY_CUSTOM_VAR=$MY_CUSTOM_VAR
export MY_CUSTOM_VAR
EOF
```

After starting the ELK services, the container will run the script at `/usr/local/bin/elk-post-hooks.sh` if it exists and is executable.

This can for instance be used to add index templates to Elasticsearch or to add index patterns to Kibana after the services have started.

Forwarding logs

Forwarding logs from a host relies on a forwarding agent that collects logs (e.g. from log files, from the syslog daemon) and sends them to our instance of Logstash.

Forwarding logs with Filebeat

Install Filebeat on the host you want to collect and forward logs from (see the *References* section for links to detailed instructions).

Note – Make sure that the version of Filebeat is the same as the version of the ELK image.

Example Filebeat set-up and configuration

Note – The `nginx-filebeat` subdirectory of the source Git repository on GitHub contains a sample `Dockerfile` which enables you to create a Docker image that implements the steps below.

Here is a sample `/etc/filebeat/filebeat.yml` configuration file for Filebeat, that forwards syslog and authentication logs, as well as nginx logs.

```
output:
  logstash:
    enabled: true
    hosts:
      - elk:5044
    ssl:
      certificate_authorities:
        - /etc/pki/tls/certs/logstash-beats.crt
    timeout: 15

filebeat:
  prospectors:
    -
      paths:
        - /var/log/syslog
        - /var/log/auth.log
      document_type: syslog
    -
      paths:
        - "/var/log/nginx/*.log"
      document_type: nginx-access
```

In the sample configuration file, make sure that you replace `elk` in `elk:5044` with the hostname or IP address of the ELK-serving host.

You'll also need to copy the `logstash-beats.crt` file (which contains the certificate authority's certificate – or server certificate as the certificate is self-signed – for Logstash's Beats input plugin; see Security considerations for more information on certificates) from the source repository of the ELK image to `/etc/pki/tls/certs/logstash-beats.crt`.

Note – Alternatively, when using Filebeat on a Windows machine, instead of using the `certificate_authorities` configuration option, the certificate from `logstash-beats.crt` can be installed in Windows' Trusted Root Certificate Authorities store.

Note – The ELK image includes configuration items (`/etc/logstash/conf.d/11-nginx.conf` and `/opt/logstash/patterns/nginx`) to parse nginx access logs, as forwarded by the Filebeat instance above.

If you're starting Filebeat for the first time, you should load the default index template in Elasticsearch. *At the time of writing, in version 6, loading the index template in Elasticsearch doesn't work, see Known issues.*

Start Filebeat:

```
    sudo /etc/init.d/filebeat start
```

Note on processing multiline log entries

In order to process multiline log entries (e.g. stack traces) as a single event using Filebeat, you may want to consider Filebeat's multiline option, which was introduced in Beats 1.1.0, as a handy alternative to altering Logstash's configuration files to use Logstash's multiline codec.

Connecting a Docker container to an ELK container running on the same host

If you want to forward logs from a Docker container to the ELK container on a host, then you need to connect the two containers.

Note – The log-emitting Docker container must have Filebeat running in it for this to work.

First of all, create an isolated, user-defined `bridge` network (we'll call it `elknet`):

```
$ sudo docker network create -d bridge elknet
```

Now start the ELK container, giving it a name (e.g. `elk`) using the `--name` option, and specifying the network it must connect to (`elknet` in this example):

```
$ sudo docker run -p 5601:5601 -p 9200:9200 -p 5044:5044 -it \
    --name elk --network=elknet sebp/elk
```

Then start the log-emitting container on the same network (replacing `your/image` with the name of the Filebeat-enabled image you're forwarding logs from):

```
$ sudo docker run -p 80:80 -it --network=elknet your/image
```

From the perspective of the log emitting container, the ELK container is now known as `elk`, which is the hostname to be used under `hosts` in the `filebeat.yml` configuration file.

For more information on networking with Docker, see Docker's documentation on working with `network` commands.

Linking containers without a user-defined network

This is the legacy way of connecting containers over the Docker's default `bridge` network, using links, which are a deprecated legacy feature of Docker which may eventually be removed.

First of all, give the ELK container a name (e.g. `elk`) using the `--name` option:

```
$ sudo docker run -p 5601:5601 -p 9200:9200 -p 5044:5044 -it --name elk sebp/elk
```

Then start the log-emitting container with the `--link` option (replacing `your/image` with the name of the Filebeat-enabled image you're forwarding logs from):

```
$ sudo docker run -p 80:80 -it --link elk:elk your/image
```

From the perspective of the log emitting container, the ELK container is now known as `elk`, which is the hostname to be used under `hosts` in the `filebeat.yml` configuration file.

With Compose here's what example entries for a (locally built log-generating) container and an ELK container might look like in the `docker-compose.yml` file.

```
yourapp:
  image: your/image
  ports:
    - "80:80"
  links:
    - elk

elk:
  image: sebp/elk
  ports:
    - "5601:5601"
    - "9200:9200"
    - "5044:5044"
```

Building the image

To build the Docker image from the source files, first clone the Git repository, go to the root of the cloned directory (i.e. the directory that contains `Dockerfile`), and:

- If you're using the vanilla `docker` command then run `sudo docker build -t <repository-name> .`, where `<repository-name>` is the repository name to be applied to the image, which you can then use to run the image with the `docker run` command.
- If you're using Compose then run `sudo docker-compose build elk`, which uses the `docker-compose.yml` file from the source repository to build the image. You can then run the built image with `sudo docker-compose up`.

Tweaking the image

There are several approaches to tweaking the image:

- Use the image as a base image and extend it, adding files (e.g. configuration files to process logs sent by log-producing applications, plugins for Elasticsearch) and overwriting files (e.g. configuration files, certificate and private key files) as required. See Docker's Dockerfile Reference page for more information on writing a `Dockerfile`.
- Replace existing files by bind-mounting local files to files in the container. See Docker's Manage data in containers page for more information on volumes in general and bind-mounting in particular.
- Fork the source Git repository and hack away.

The next few subsections present some typical use cases.

Updating Logstash's configuration

The image contains several configuration files for Logstash (e.g. `01-lumberjack-input.conf`, `02-beats-input.conf`), all located in `/etc/logstash/conf.d`.

To modify an existing configuration file, you can bind-mount a local configuration file to a configuration file within the container at runtime. For instance, if you want to replace the image's `30-output.conf` Logstash configuration file with your local file `/path/to/your-30-output.conf`, then you would add the following `-v` option to your `docker` command line:

```
$ sudo docker run ... \
    -v /path/to/your-30-output.conf:/etc/logstash/conf.d/30-output.conf \
    ...
```

To create your own image with updated or additional configuration files, you can create a `Dockerfile` that extends the original image, with contents such as the following:

```
FROM sebp/elk

# overwrite existing file
ADD /path/to/your-30-output.conf /etc/logstash/conf.d/30-output.conf

# add new file
ADD /path/to/new-12-some-filter.conf /etc/logstash/conf.d/12-some-filter.conf
```

Then build the extended image using the `docker build` syntax.

Installing Elasticsearch plugins

Elasticsearch's home directory in the image is `/opt/elasticsearch`, its plugin management script (`elasticsearch-plugin`) resides in the `bin` subdirectory, and plugins are installed in `plugins`.

Elasticsearch runs as the user `elasticsearch`. To avoid issues with permissions, it is therefore recommended to install Elasticsearch plugins as `elasticsearch`, using the `gosu` command (see below for an example, and references for further details).

A `Dockerfile` like the following will extend the base image and install the GeoIP processor plugin (which adds information about the geographical location of IP addresses):

```
FROM sebp/elk

ENV ES_HOME /opt/elasticsearch
WORKDIR ${ES_HOME}

RUN CONF_DIR=/etc/elasticsearch gosu elasticsearch bin/elasticsearch-plugin \
    install ingest-geoip
```

You can now build the new image (see the *Building the image* section above) and run the container in the same way as you did with the base image.

Installing Logstash plugins

The name of Logstash's home directory in the image is stored in the `LOGSTASH_HOME` environment variable (which is set to `/opt/logstash` in the base image). Logstash's plugin management script (`logstash-plugin`) is located in the `bin` subdirectory.

Logstash runs as the user `logstash`. To avoid issues with permissions, it is therefore recommended to install Logstash plugins as `logstash`, using the `gosu` command (see below for an example, and references for further details).

The following `Dockerfile` can be used to extend the base image and install the RSS input plugin:

```
FROM sebp/elk

WORKDIR ${LOGSTASH_HOME}
RUN gosu logstash bin/logstash-plugin install logstash-input-rss
```

See the *Building the image* section above for instructions on building the new image. You can then run a container based on this image using the same command line as the one in the *Usage* section.

Installing Kibana plugins

The name of Kibana's home directory in the image is stored in the `KIBANA_HOME` environment variable (which is set to `/opt/kibana` in the base image). Kibana's plugin management script (`kibana-plugin`) is located in the `bin` subdirectory, and plugins are installed in `installedPlugins`.

Kibana runs as the user `kibana`. To avoid issues with permissions, it is therefore recommended to install Kibana plugins as `kibana`, using the `gosu` command (see below for an example, and references for further details).

The following `Dockerfile` can be used to extend the base image and install the latest version of the Sense plugin, a handy console for interacting with the REST API of Elasticsearch:

```
FROM sebp/elk

WORKDIR ${KIBANA_HOME}
RUN gosu kibana bin/kibana-plugin install elastic/sense
```

See the *Building the image* section above for instructions on building the new image. You can then run a container based on this image using the same command line as the one in the *Usage* section. The Sense interface will be accessible at `http://<your-host>:5601/apss/sense` (e.g. http://localhost:5601/app/sense for a local native instance of Docker).

Persisting log data

In order to keep log data across container restarts, this image mounts `/var/lib/elasticsearch` — which is the directory that Elasticsearch stores its data in — as a volume.

You may however want to use a dedicated data volume to persist this log data, for instance to facilitate back-up and restore operations.

One way to do this is to mount a Docker named volume using `docker`'s `-v` option, as in:

```
$ sudo docker run -p 5601:5601 -p 9200:9200  -p 5044:5044 \
    -v elk-data:/var/lib/elasticsearch --name elk sebp/elk
```

This command mounts the named volume `elk-data` to `/var/lib/elasticsearch` (and automatically creates the volume if it doesn't exist; you could also pre-create it manually using `docker volume create elk-data`).

Note – By design, Docker never deletes a volume automatically (e.g. when no longer used by any container). Whilst this avoids accidental data loss, it also means that things can become messy if you're not managing your volumes properly (e.g. using the `-v` option when removing containers with `docker rm` to also delete the volumes... bearing in mind that the actual volume won't be deleted as long as at least one container is still referencing it, even if it's not running). You can keep track of existing volumes using `docker volume ls`.

See Docker's page on Managing Data in Containers and Container42's Docker In-depth: Volumes page for more information on managing data volumes.

In terms of permissions, Elasticsearch data is created by the image's `elasticsearch` user, with UID 991 and GID 991.

There is a known situation where SELinux denies access to the mounted volume when running in *enforcing* mode. The workaround is to use the `setenforce 0` command to run SELinux in *permissive* mode.

Setting up an Elasticsearch cluster

The ELK image can be used to run an Elasticsearch cluster, either on separate hosts or (mainly for test purposes) on a single host, as described below.

For more (non-Docker-specific) information on setting up an Elasticsearch cluster, see the Life Inside a Cluster section section of the Elasticsearch definitive guide.

Running Elasticsearch nodes on different hosts

To run cluster nodes on different hosts, you'll need to update Elasticsearch's `/etc/elasticsearch/elasticsearch.yml` file in the Docker image so that the nodes can find each other:

- Configure the zen discovery module, by adding a `discovery.zen.ping.unicast.hosts` directive to point to the IP addresses or hostnames of hosts that should be polled to perform discovery when Elasticsearch is started on each node.

- Set up the `network.*` directives as follows:

  ```
  network.host: 0.0.0.0

  network.publish_host: <reachable IP address or FQDN>
  ```

  where `reachable IP address` refers to an IP address that other nodes can reach (e.g. a public IP address, or a routed private IP address, but *not* the Docker-assigned internal 172.x.x.x address).
- Publish port 9300

As an example, start an ELK container as usual on one host, which will act as the first master. Let's assume that the host is called *elk-master.example.com*.

Have a look at the cluster's health:

```
$ curl http://elk-master.example.com:9200/_cluster/health?pretty
{
  "cluster_name" : "elasticsearch",
  "status" : "yellow",
  "timed_out" : false,
  "number_of_nodes" : 1,
  "number_of_data_nodes" : 1,
  "active_primary_shards" : 6,
  "active_shards" : 6,
  "relocating_shards" : 0,
  "initializing_shards" : 0,
  "unassigned_shards" : 6,
  "delayed_unassigned_shards" : 6,
  "number_of_pending_tasks" : 0,
  "number_of_in_flight_fetch" : 0,
  "task_max_waiting_in_queue_millis" : 0,
  "active_shards_percent_as_number" : 50.0
}
```

This shows that only one node is up at the moment, and the `yellow` status indicates that all primary shards are active, but not all replica shards are active.

Then, on another host, create a file named `elasticsearch-slave.yml` (let's say it's in `/home/elk`), with the following contents:

```
network.host: 0.0.0.0
network.publish_host: <reachable IP address or FQDN>
discovery.zen.ping.unicast.hosts: ["elk-master.example.com"]
```

You can now start an ELK container that uses this configuration file, using the following command (which mounts the configuration files on the host into the container):

```
$ sudo docker run -it --rm=true -p 9200:9200 -p 9300:9300 \
  -v /home/elk/elasticsearch-slave.yml:/etc/elasticsearch/elasticsearch.yml \
  sebp/elk
```

Once Elasticsearch is up, displaying the cluster's health on the original host now shows:

```
$ curl http://elk-master.example.com:9200/_cluster/health?pretty
{
  "cluster_name" : "elasticsearch",
  "status" : "green",
  "timed_out" : false,
  "number_of_nodes" : 2,
  "number_of_data_nodes" : 2,
  "active_primary_shards" : 6,
  "active_shards" : 12,
  "relocating_shards" : 0,
  "initializing_shards" : 0,
  "unassigned_shards" : 0,
  "delayed_unassigned_shards" : 0,
  "number_of_pending_tasks" : 0,
  "number_of_in_flight_fetch" : 0,
  "task_max_waiting_in_queue_millis" : 0,
  "active_shards_percent_as_number" : 100.0
}
```

Running Elasticsearch nodes on a single host

Setting up Elasticsearch nodes to run on a single host is similar to running the nodes on different hosts, but the containers need to be linked in order for the nodes to discover each other.

Start the first node using the usual `docker` command on the host:

```
$ sudo docker run -p 5601:5601 -p 9200:9200 -p 5044:5044 -it --name elk sebp/elk
```

Now, create a basic `elasticsearch-slave.yml` file containing the following lines:

```
network.host: 0.0.0.0
discovery.zen.ping.unicast.hosts: ["elk"]
```

Start a node using the following command:

```
$ sudo docker run -it --rm=true \
  -v /var/sandbox/elk-docker/elasticsearch-slave.yml:/etc/elasticsearch/elasticsearch.yml \
  --link elk:elk --name elk-slave sebp/elk
```

Note that Elasticsearch's port is not published to the host's port 9200, as it was already published by the initial ELK container.

Optimising your Elasticsearch cluster

You can use the ELK image as is to run an Elasticsearch cluster, especially if you're just testing, but to optimise your set-up, you may want to have:

- One node running the complete ELK stack, using the ELK image as is.
- Several nodes running *only* Elasticsearch (see *Starting services selectively*).

An even more optimal way to distribute Elasticsearch, Logstash and Kibana across several nodes or hosts would be to run only the required services on the appropriate nodes or hosts (e.g. Elasticsearch on several hosts, Logstash on a dedicated host, and Kibana on another dedicated host).

Security considerations

As it stands this image is meant for local test use, and as such hasn't been secured: access to the ELK services is unrestricted, and default authentication server certificates and private keys for the Logstash input plugins are bundled with the image.

To harden this image, at the very least you would want to:

- Restrict the access to the ELK services to authorised hosts/networks only, as described in e.g. Elasticsearch Scripting and Security and Elastic Security: Deploying Logstash, ElasticSearch, Kibana "securely" on the Internet.
- Password-protect the access to Kibana and Elasticsearch (see SSL And Password Protection for Kibana).
- Generate a new self-signed authentication certificate for the Logstash input plugins (see Notes on certificates) or (better) get a proper certificate from a commercial provider (known as a certificate authority), and keep the private key private.

The sebp/elkx image, which extends the ELK image with X-Pack, may be a useful starting point to improve the security of the ELK services.

If on the other hand you want to disable certificate-based server authentication (e.g. in a demo environment), see Disabling SSL/TLS.

Notes on certificates

Dummy server authentication certificates (`/etc/pki/tls/certs/logstash-*.crt`) and private keys (`/etc/pki/tls/private/logstash-*.key`) are included in the image.

Note – For Logstash 2.4.0 a PKCS#8-formatted private key must be used (see *Breaking changes* for guidance).

The certificates are assigned to hostname `*`, which means that they will work if you are using a single-part (i.e. no dots) domain name to reference the server from your client.

Example – In your client (e.g. Filebeat), sending logs to hostname `elk` will work, `elk.mydomain.com` will not (will produce an error along the lines of `x509: certificate is valid for *, not elk.mydomain.com`), neither will an IP address such as 192.168.0.1 (expect `x509: cannot validate certificate for 192.168.0.1 because it doesn't contain any IP SANs`).

If you cannot use a single-part domain name, then you could consider:

- Issuing a self-signed certificate with the right hostname using a variant of the commands given below.
- Issuing a certificate with the IP address of the ELK stack in the subject alternative name field, even though this is bad practice in general as IP addresses are likely to change.
- Adding a single-part hostname (e.g. `elk`) to your client's `/etc/hosts` file.

The following commands will generate a private key and a 10-year self-signed certificate issued to a server with hostname `elk` for the Beats input plugin:

```
$ cd /etc/pki/tls
$ sudo openssl req -x509 -batch -nodes -subj "/CN=elk/" \
    -days 3650 -newkey rsa:2048 \
    -keyout private/logstash-beats.key -out certs/logstash-beats.crt
```

As another example, when running a non-predefined number of containers concurrently in a cluster with hostnames *directly* under the `.mydomain.com` domain (e.g. `elk1.mydomain.com`, `elk2.mydomain.com`, etc.; *not* `elk1.subdomain.mydomain.com`, `elk2.othersubdomain.mydomain.com` etc.), you could create a certificate assigned to the wildcard hostname `*.example.com` by using the following command (all other parameters are identical to the ones in the previous example).

```
$ cd /etc/pki/tls
$ sudo openssl req -x509 -batch -nodes -subj "/CN=*.example.com/" \
    -days 3650 -newkey rsa:2048 \
    -keyout private/logstash-beats.key -out certs/logstash-beats.crt
```

To make Logstash use the generated certificate to authenticate to a Beats client, extend the ELK image to overwrite (e.g. using the `Dockerfile` directive `ADD`):

- the certificate file (`logstash-beats.crt`) with `/etc/pki/tls/certs/logstash-beats.crt`.
- the private key file (`logstash-beats.key`) with `/etc/pki/tls/private/logstash-beats.key`,

Additionally, remember to configure your Beats client to trust the newly created certificate using the `certificate_authorities` directive, as presented in Forwarding logs with Filebeat.

Disabling SSL/TLS

Certificate-based server authentication requires log-producing clients to trust the server's root certificate authority's certificate, which can be an unnecessary hassle in zero-criticality environments (e.g. demo environments, sandboxes).

To disable certificate-based server authentication, remove all `ssl` and `ssl`-prefixed directives (e.g. `ssl_certificate`, `ssl_key`) in Logstash's input plugin configuration files.

For instance, with the default configuration files in the image, replace the contents of `02-beats-input.conf` (for Beats emitters) with:

```
input {
  beats {
    port => 5044
  }
}
```

Frequently encountered issues

Elasticsearch is not starting (1):

`max virtual memory areas vm.max_map_count [65530] likely too low, increase to at least [262144`

If the container stops and its logs include the message `max virtual memory areas vm.max_map_count [65530] likely too low, increase to at least [262144]`, then the limits on mmap counts are too low, see Prerequisites.

Elasticsearch is not starting (2):

`cat: /var/log/elasticsearch/elasticsearch.log: No such file or directory`

If Elasticsearch's logs are *not* dumped (i.e. you get the following message: `cat: /var/log/elasticsearch/elasticsearch.log: No such file or directory`), then Elasticsearch did not have enough memory to start, see Prerequisites.

Elasticsearch is not starting (3): bootstrap tests

As from version 5, if Elasticsearch is no longer starting, i.e. the `waiting for Elasticsearch to be up (xx/30)` counter goes up to 30 and the container exits with `Couln't start Elasticsearch. Exiting.` *and* Elasticsearch's logs are dumped, then read the recommendations in the logs and consider that they *must* be applied.

In particular, in case (1) above, the message `max virtual memory areas vm.max_map_count [65530] likely too low, increase to at least [262144]` means that the host's limits on mmap counts **must** be set to at least 262144.

Elasticsearch is suddenly stopping after having started properly

With the default image, this is usually due to Elasticsearch running out of memory after the other services are started, and the corresponding process being (silently) killed.

As a reminder (see Prerequisites), you should use no less than 3GB of memory to run the container... and possibly much more.

Known issues

When using Filebeat, an index template file is used to connect to Elasticsearch to define settings and mappings that determine how fields should be analysed.

In version 5, before starting Filebeat for the first time, you would run this command (replacing `elk` with the appropriate hostname) to load the default index template in Elasticsearch:

```
curl -XPUT 'http://elk:9200/_template/filebeat?pretty' -d@/etc/filebeat/filebeat.template.json
```

In version 6 however, the `filebeat.template.json` template file has been replaced with a `fields.yml` file, which is used to load the index manually by running `filebeat setup --template` as per the official Filebeat instructions. Unfortunately, this doesn't currently work and results in the following message:

```
Exiting: Template loading requested but the Elasticsearch output is not configured/enabled
```

Attempting to start Filebeat without setting up the template produces the following message:

```
Warning: Couldn't read data from file "/etc/filebeat/filebeat.template.json",
Warning: this makes an empty POST.
{
  "error" : {
    "root_cause" : [
      {
        "type" : "parse_exception",
        "reason" : "request body is required"
      }
    ],
    "type" : "parse_exception",
    "reason" : "request body is required"
  },
  "status" : 400
}
```

One can assume that in later releases of Filebeat the instructions will be clarified to specify how to manually load the index template into an specific instance of Elastisearch, and that the warning message will vanish as no longer applicable in version 6.

Troubleshooting

Important – If you need help to troubleshoot the configuration of Elasticsearch, Logstash, or Kibana, regardless of where the services are running (in a Docker container or not), please head over to the Elastic forums. The troubleshooting guidelines below only apply to running a container using the ELK Docker image.

Here are a few pointers to help you troubleshoot your containerised ELK.

If Elasticsearch isn't starting...

If the suggestions listed in Frequently encountered issues don't help, then an additional way of working out why Elasticsearch isn't starting is to:

- Start a container with the `bash` command:

  ```
  $ sudo docker run -it docker_elk bash
  ```

- Start Elasticsearch manually to look at what it outputs:

  ```
  $ gosu elasticsearch /opt/elasticsearch/bin/elasticsearch \
      -Edefault.path.logs=/var/log/elasticsearch \
      -Edefault.path.data=/var/lib/elasticsearch \
      -Edefault.path.conf=/etc/elasticsearch
  ```

If your log-emitting client doesn't seem to be able to reach Logstash...

Note – Similar troubleshooting steps are applicable in set-ups where logs are sent directly to Elasticsearch.

Make sure that:

- You started the container with the right ports open (e.g. 5044 for Beats).
- If you are using Filebeat, its version is the same as the version of the ELK image/stack.
- The ports are reachable from the client machine (e.g. make sure the appropriate rules have been set up on your firewalls to authorise outbound flows from your client and inbound flows on your ELK-hosting machine).
- Your client is configured to connect to Logstash using TLS (or SSL) and that it trusts Logstash's self-signed certificate (or certificate authority if you replaced the default certificate with a proper certificate – see Security considerations).

  To check if Logstash is authenticating using the right certificate, check for errors in the output of

  ```
  $ openssl s_client -connect localhost:5044 -CAfile logstash-beats.crt
  ```

  where `logstash-beats.crt` is the name of the file containing Logstash's self-signed certificate.

Additional tips

If the suggestions given above don't solve your issue, then you should have a look at:

- Your log-emitting client's logs.
- ELK's logs, by `docker exec`'ing into the running container (see Creating a dummy log entry), turning on stdout log (see plugins-outputs-stdout), and checking Logstash's logs (located in `/var/log/logstash`), Elasticsearch's logs (in `/var/log/elasticsearch`), and Kibana's logs (in `/var/log/kibana`).

  Note that ELK's logs are rotated daily and are deleted after a week, using logrotate. You can change this behaviour by overwriting the `elasticsearch`, `logstash` and `kibana` files in `/etc/logrotate.d`.

Reporting issues

Important – For *non-Docker-related* issues with Elasticsearch, Kibana, and Elasticsearch, report the issues on the appropriate Elasticsearch, Logstash, or Kibana GitHub repository.

You can report issues with this image using GitHub's issue tracker (please avoid raising issues as comments on Docker Hub, if only for the fact that the notification system is broken at the time of writing so there's a fair chance that I won't see it for a while).

Bearing in mind that the first thing I'll need to do is reproduce your issue, please provide as much relevant information (e.g. logs, configuration files, what you were expecting and what you got instead, any troubleshooting steps that you took, what *is* working) as possible for me to do that.

Pull requests are also welcome if you have found an issue and can solve it.

Breaking changes

Here is the list of breaking changes that may have side effects when upgrading to later versions of the ELK image:

- **Version 6**

  Applies to tags: `600` *and later.*

  Breaking changes are introduced in version 6 of Elasticsearch, Logstash, and Kibana.
- `ES_HEAP_SIZE` **and** `LS_HEAP_SIZE`

  Applies to tags: `502` *to* `522`.

  Overriding the `ES_HEAP_SIZE` and `LS_HEAP_SIZE` environment variables has no effect on the heap size used by Elasticsearch and Logstash (see issue #129).
- **Elasticsearch home directory**

  Applies to tags: `502` *and later.*

  Elasticsearch is no longer installed from the `deb` package (which attempts, in version 5.0.2, to modify system files that aren't accessible from a container); instead it is installed from the `tar.gz` package.

  As a consequence, Elasticsearch's home directory is now `/opt/elasticsearch` (was `/usr/share/elasticsearch`).
- **Version 5**

  Applies to tags: `es500_l500_k500` *and later.*

  Breaking changes are introduced in version 5 of Elasticsearch, Logstash, and Kibana.
- **Private keys in PKCS#8 format**

  Applies to tags: `es240_l240_k460` *and* `es241_l240_k461`.

  In Logstash version 2.4.x, the private keys used by Logstash with the Beats input are expected to be in PKCS#8 format. To convert the private key (`logstash-beats.key`) from its default PKCS#1 format to PKCS#8, use the following command:

  ```
  $ openssl pkcs8 -in logstash-beats.key -topk8 -nocrypt -out logstash-beats.p8
  ```

  and point to the `logstash-beats.p8` file in the `ssl_key` option of Logstash's `02-beats-input.conf` configuration file.
- **Logstash forwarder**

  Applies to tags: `es500_l500_k500` *and later.*

  The use of Logstash forwarder is deprecated, its Logstash input plugin configuration has been removed, and port 5000 is no longer exposed.
- **UIDs and GIDs**

Applies to tags: `es235_l234_k454` *and later.*

Fixed UIDs and GIDs are now assigned to Elasticsearch (both the UID and GID are 991), Logstash (992), and Kibana (993).

- **Java 8**

  Applies to tags: `es234_l234_k452` *and later.*

  This image initially used Oracle JDK 7, which is no longer updated by Oracle, and no longer available as a Ubuntu package.

  As from tag `es234_l234_k452`, the image uses Oracle JDK 8. This may have unintended side effects on plugins that rely on Java.

- **Logstash configuration auto-reload**

  Applies to tags: `es231_l231_k450`, `es232_l232_k450`.

  Logstash's configuration auto-reload option was introduced in Logstash 2.3 and enabled in the images with tags `es231_l231_k450` and `es232_l232_k450`.

  As this feature created a resource leak prior to Logstash 2.3.3 (see https://github.com/elastic/logstash/issues/5235), the `--auto-reload` option was removed as from the `es233_l232_k451`-tagged image (see https://github.com/spujadas/elk-docker/issues/41).

  Users of images with tags `es231_l231_k450` and `es232_l232_k450` are strongly recommended to override Logstash's options to disable the auto-reload feature by setting the `LS_OPTS` environment variable to `--no-auto-reload` if this feature is not needed.

  To enable auto-reload in later versions of the image:

  - From `es500_l500_k500` onwards: add the `--config.reload.automatic` command-line option to `LS_OPTS`.
  - From `es234_l234_k452` to `es241_l240_k461`: add `--auto-reload` to `LS_OPTS`.

References

- How To Install Elasticsearch, Logstash, and Kibana 4 on Ubuntu 14.04
- The Docker Book
- The Logstash Book
- Elastic's reference documentation:
  - Elasticsearch Reference
  - Logstash Reference
  - Kibana Reference
  - Filebeat Reference
- gosu, simple Go-based setuid+setgid+setgroups+exec, a convenient alternative to `USER` in Dockerfiles (see Best practices for writing Dockerfiles)

About