Getting started with the Elastic Stack and Docker Compose: Part 1

By **Eddie Mitchell**

17 May 2023

As the Elastic Stack has grown over the years and the feature sets have increased, so has the complexity of getting started or attempting a proof-of-concept (POC) locally. And while **Elastic Cloud** is still the fastest and easiest way to get started with Elastic, the need for local development and testing is still widely abundant. As developers, we are drawn to quick setups and rapid development with low-effort results. Nothing screams fast setup and POC quite like Docker — which is what we'll be focusing on to get started with an entire Elastic Stack build-out for your local enjoyment.

In part one of this two-part series, we'll dive into configuring the components of a standard Elastic Stack consisting of Elasticsearch, Logstash, Kibana, and Beats (ELK-B), on which we can immediately begin developing.

In part two, we'll enhance our base configuration and add many of the different features that power our evolving stack, such as APM, Agent, Fleet, Integrations, and Enterprise Search. We will also look at instrumenting these in our new local environment for development and POC purposes.

For those who have been through some of this before, you're welcome to **TL;DR and head over to the repo to grab the files**.

As a prerequisite, **Docker Desktop or Docker Engine** with **Docker-Compose** will need to be installed and configured. For this tutorial, we will be using Docker Desktop.

Our focus for these Docker containers will primarily be Elasticsearch and Kibana. However, we'll be utilizing Metricbeat to give us some cluster insight as well as Filebeat and Logstash for some ingestion basics.

File structure

First, let's start by defining the outline of our file structure.

```
├── .env
```

```
├── docker-compose.yml
├── filebeat.yml
├── logstash.conf
└── metricbeat.yml
```

We'll keep it simple initially. Elasticsearch and Kibana will be able to start from the docker-compose file, while Filebeat, Metricbeat, and Logstash will all need additional configuration from yml files.

Environment file

Next, we'll define variables to pass to the docker-compose via the .env file. These parameters will help us establish ports, memory limits, component versions, etc.

.env

```
# Project namespace (defaults to the current fo
#COMPOSE_PROJECT_NAME=myproject


# Password for the 'elastic' user (at least 6 c
ELASTIC_PASSWORD=changeme


# Password for the 'kibana_system' user (at lea
KIBANA_PASSWORD=changeme
```

Copy

Note that the placeholder word “changeme” for all the passwords and the sample key are used for demonstration purposes only. These should be changed even for your local POC needs.

As you can see here, we specify ports 9200 and 5601 for Elasticsearch and Kibana respectively. This is also where you can change from “basic” to “trial” license type in order to test additional features.

We make use of the `STACK_VERSION' environment variable here in order to pass it to each of the services (containers) in our docker-compose.yml file. When using Docker, opting to hard-code the version number as opposed to using something like the `:latest` tag is a good way to maintain positive control over the environment. For components of the Elastic Stack, the `:latest` tag is not supported and we require version numbers to pull the images.

Setup and Elasticsearch node

One of the first bits of trouble that's often run into when getting started is security configuration. As of 8.0, security is enabled by default. Therefore, we'll need to make sure we have the certificate CA setup correctly by utilizing a "setup" node to establish the certificates. Having **security enabled is a recommended practice** and should not be disabled, even in POC environments.

docker-compose.yml (‘setup’ container)

```
version: "3.8"


volumes:
 certs:
   driver: local
 esdata01:
```

```
    driver: local
  kibanadata:
    driver: local
  metricbeatdata01:
```

Copy

At the top of the docker-compose.yml we set the compose versio followed by the volumes and default networking configuration that will be used throughout our different containers.

We also see that we're standing up a container labeled “setup” wi call the elasticsearch-certutil, passing the server names in yml format in order to create the CA cert and node certs. If you wanted to have more than one Elasticsearch node in your stack, this is where you would add the server name to allow the cert creation.

Note: In a future post, we’ll adopt the recommended method of using a keystore to keep secrets, but for now, this will allow us to get the cluster up and running.

This setup container will start up first, wait for the ES01 container to come online, and then use our environment variables to set up the passwords we want in our cluster. We’re also saving all certificates to the “certs” volume so that all other containers can have access to them.

Since the Setup container is dependent on the ES01 container, let's take a quick look at the next configuration so we can start them both up:

docker-compose.yml (‘es01’container)

```
  es01:
    depends_on:
      setup:
```

```
      condition: service_healthy
  image: docker.elastic.co/elasticsearch/elast
  labels:
    co.elastic.logs/module: elasticsearch
  volumes:
    - certs:/usr/share/elasticsearch/config/ce
    - esdata01:/usr/share/elasticsearch/data
  ports:
    - ${ES_PORT}:9200
```

Copy

This will be the single-node cluster of Elasticsearch that we're using for testing.

Notice we'll be using the CA cert and node certificates that were generated.

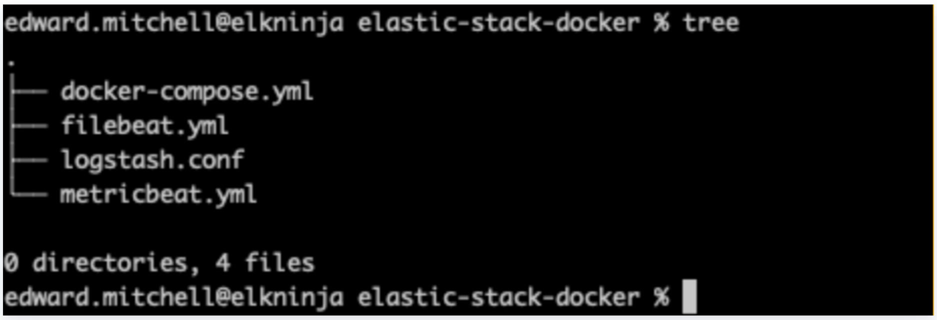
You will also notice that we're storing the Elasticsearch data in a volume outside of the container by specifying `- esdata01:/usr/share/elasticsearch/data` The two primary reasons for this are performance and data persistence. If we were to leave the data directory inside the container, we would see a significant degradation in the performance of our Elasticsearch node, as well as lose data anytime we needed to change the configuration of the container within our docker-compose file.

With both configurations in place, we can perform our first `docker-compose up` command.

Docker Compose tips

If you're new to Docker Compose or it's been a while since you've had to **remember some of the commands**, let's quickly review the primary ones you will want to know for this adventure.

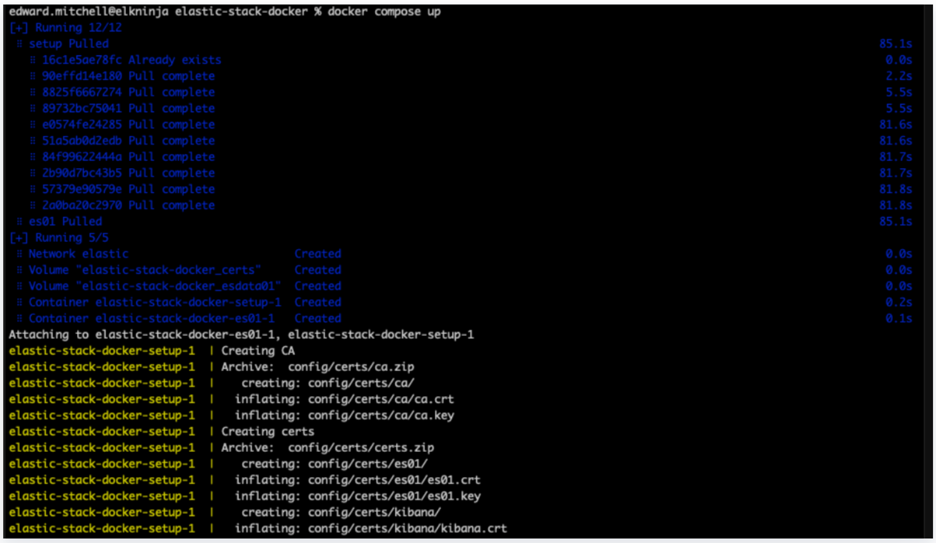
You will want to run all these commands in a terminal while in the same folder in which your docker-compose.yml file resides. My example folder:



Let’s take a look at those commands.

`docker compose up`	Creates and starts all containers within the docker-compose.yml
`docker compose down`	Stops all containers and removes any networks relating to the docker-compose.yml. Leaves Volumes that were created intact and allows data that was generated to persist between builds of the environment.
`docker compose down -v`	Same as above, except also removes all volumes that were created. This is great for “starting over.”
`docker compose start`	If the containers exist but are stopped, this will start all the containers.
`docker compose stop`	Stops all containers and leaves the environment intact. If you are in a running terminal, then `CTRL+C` will also exit and stop all containers.

Now, lets run `docker-compose up`.



At this point, if the syntax is correct, Docker will begin to download all images and build the environment that is listed in the docker-compose.yml file. This may take a few minutes depending on the speed of your internet. If you want to see the images outside of

Docker Desktop, you can always find them in the **official Elastic Docker Hub**.

Troubleshooting Virtual Memory misconfigurations

When starting up the Elasticsearch node for the first time, many users get stuck on the Virtual Memory configuration and receive an error message such as:

```
{"@timestamp":"2023-04-14T13:16:22.148Z", "log.l
```

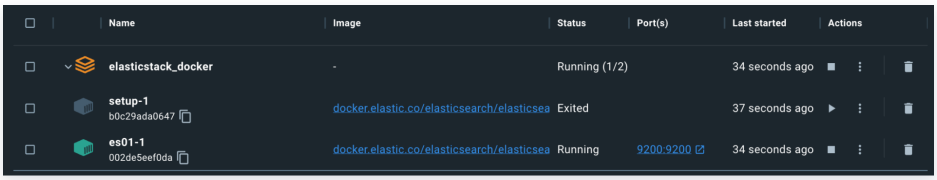
Copy

The key takeaway here is `max virtual memory areas vm.max_map_count [65530] is too low, increase to at least [262144].`

Ultimately, the command `sysctl -w vm.max_map_count=262144` needs to be run where the containers are being hosted.

In the case of Mac, **check these instructions** for Docker for Mac. Follow **these instructions for Docker Desktop**. For Linux users, **see these instructions**. Windows users, if you have Docker Desktop, **you can try these instructions**. However, if you're **using WSLv2 with Docker Desktop, take a look here**.

Once complete, you can reboot Docker Desktop and retry your `docker-compose up` command.



Remember, the Setup container will exit on purpose after it has completed generating the certs and passwords.

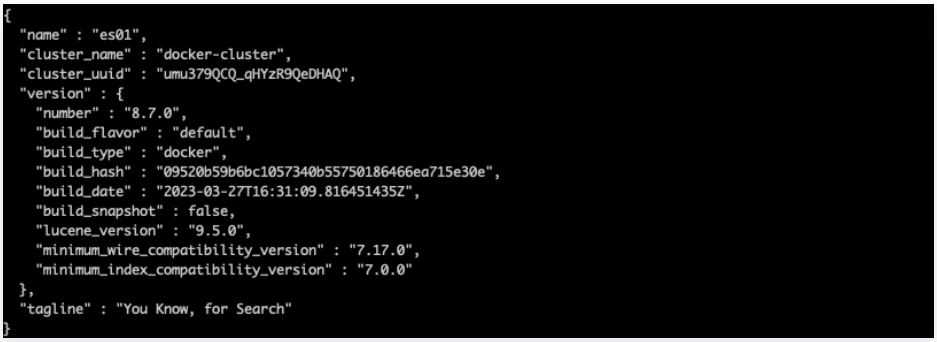
So far so good, but let's test.

We can use a command to copy the ca.crt out of the es01-1 container. Remember, the name of the set of containers is based on the folder from which the docker-compose.yml is running. For example, my directory is “elasticstack_docker” therefore, my command would look like this, based on the screenshot above:

```
docker cp

elasticstack_docker-es01-
1:/usr/share/elasticsearch/config/certs/ca/ca.crt /tmp/.
```

Once the certificate is downloaded, run a curl command to query the Elasticsearch node:

```
curl --cacert /tmp/ca.crt -u elastic:changeme
https://localhost:9200
```



Success!

Notice that we’re accessing Elasticsearch using **localhost**:9200. This is thanks to the port, which has been specified via the `ports` section of docker-compose.yml. This setting maps ports on the container to ports on the host and allows traffic to pass through your machine and into the docker container with that port specified.

Kibana

For the Kibana config, we will utilize the certificate output from earlier. We will also specify that this node doesn't start until it sees that the Elasticsearch node above is up and running correctly.

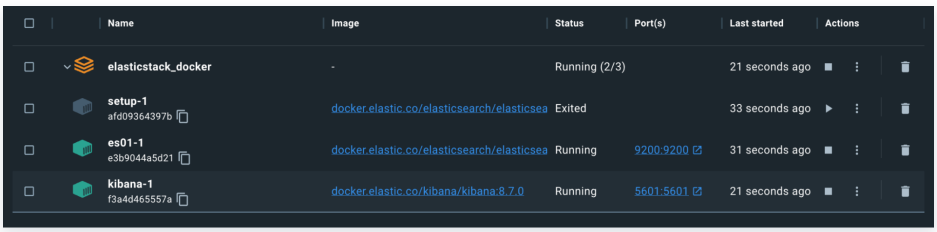
docker-compose.yml ('kibana' container)

```
kibana:
  depends_on:
    es01:
      condition: service_healthy
  image: docker.elastic.co/kibana/kibana:${STA
  labels:
    co.elastic.logs/module: kibana
  volumes:
    - certs:/usr/share/kibana/config/certs
    - kibanadata:/usr/share/kibana/data
  ports:
    - ${KIBANA_PORT}:5601
```

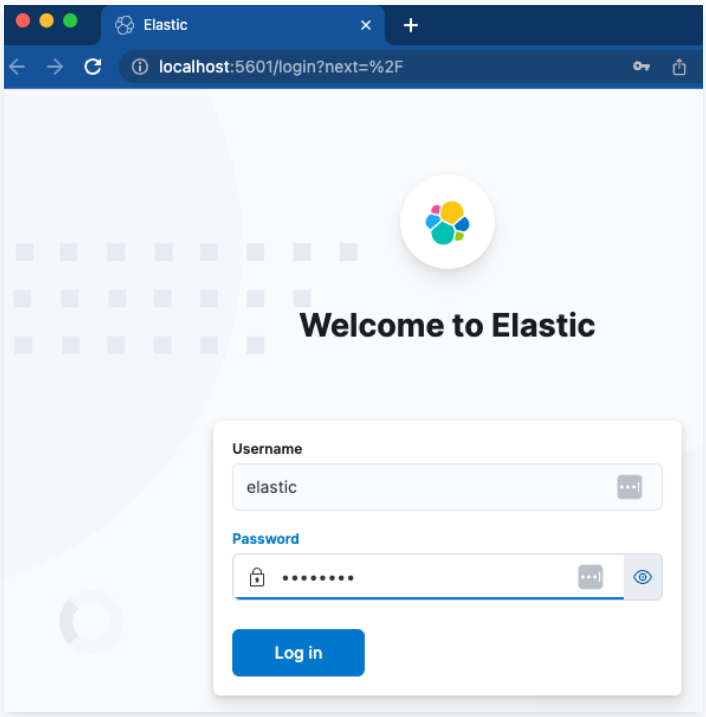
Copy

Notice in our `environment` section that we're specifying `ELASTICSEARCH_HOSTS=https://es01:9200` We're able to specify the container name here for our ES01 Elasticsearch container since we're utilizing the **Docker default networking**. All containers that are using the "elastic" network that was specified at the beginning of our docker-compose.yml file will be able to properly resolve other container names and communicate with each other.

Let's load up Kibana and see if we can access it.



The containers are green. We should now be able to reach http://localhost:5601.



A quick login with the username and password that was specified should drop us right into a brand-new instance of Kibana. Excellent!

Metricbeat

Now that we have Kibana and Elasticsearch up and running and communicating, let's configure Metricbeat to help us keep an eye on things. This will require both configuration in our docker-compose file, and also in a standalone metricbeat.yml file.

Note: For Logstash, Filebeat, and Metricbeat, the configuration files are using **bind mounts**. Bind mounts for files will retain the same permissions and ownership within the container that they have on the host system. Be sure to set permissions such that the files will be readable and, ideally, not writeable by the container's user. You will receive an error in the container otherwise. Removing the write permissions on your host may suffice.

docker-compose.yml ('metricbeat01' container)

```
 metricbeat01:
   depends_on:
     es01:
       condition: service_healthy
     kibana:
       condition: service_healthy
   image: docker.elastic.co/beats/metricbeat:$
   user: root
   volumes:
     - certs:/usr/share/metricbeat/certs
     - metricbeatdata01:/usr/share/metricbeat/
     - "./metricbeat.yml:/usr/share/metricbeat/
```

Copy

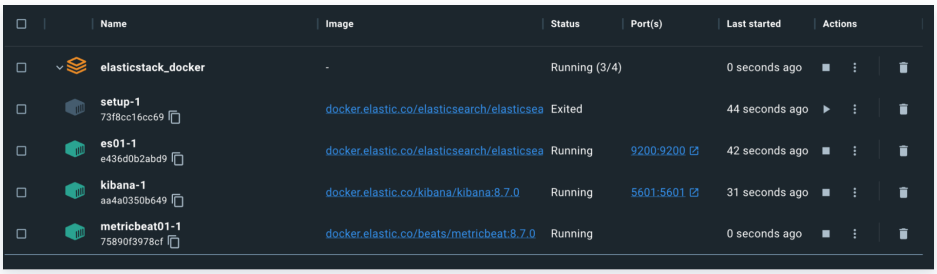
Here, we're exposing host information regarding processes, filesystem, and the docker daemon to the Metricbeat container in a read-only fashion. This enables Metricbeat to collect the data to send to Elasticsearch.

metricbeat.yml

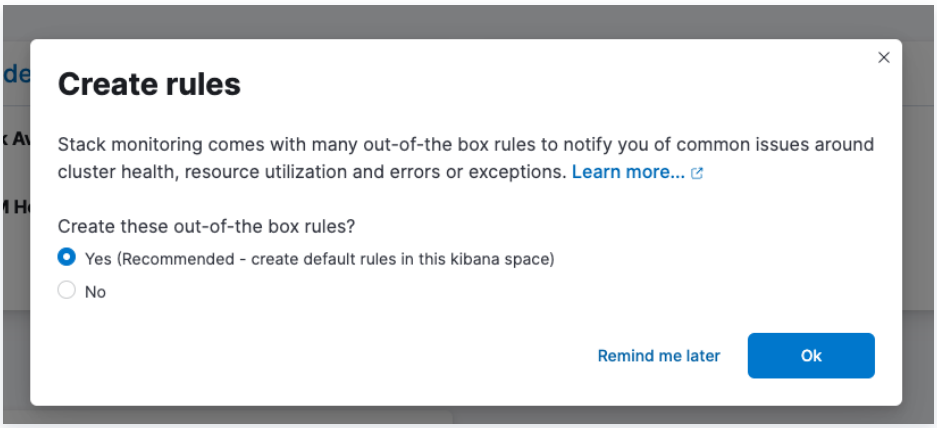
```
metricbeat.config.modules:
 path: ${path.config}/modules.d/*.yml
 reload.enabled: false


metricbeat.modules:
- module: elasticsearch
 xpack.enabled: true
 period: 10s
 hosts: ${ELASTIC_HOSTS}
 ssl.certificate_authorities: "certs/ca/ca.crt
 ssl.certificate: "certs/es01/es01.crt"
```

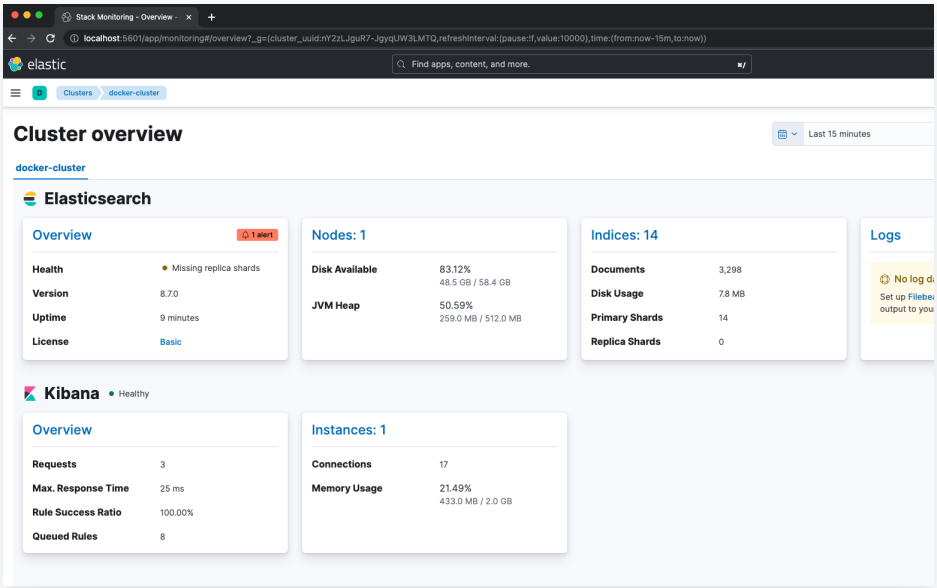
Copy

Our Metricbeat is dependent on ES01 and Kibana nodes being healthy before starting. The notable configurations here are in the metricbeat.yml file. We have enabled four modules for gathering metrics including Elasticsearch, Kibana, Logstash, and Docker. This means, once we verify Metricbeat is up, we can hop into Kibana and **navigate to "Stack Monitoring"** to see how things look.

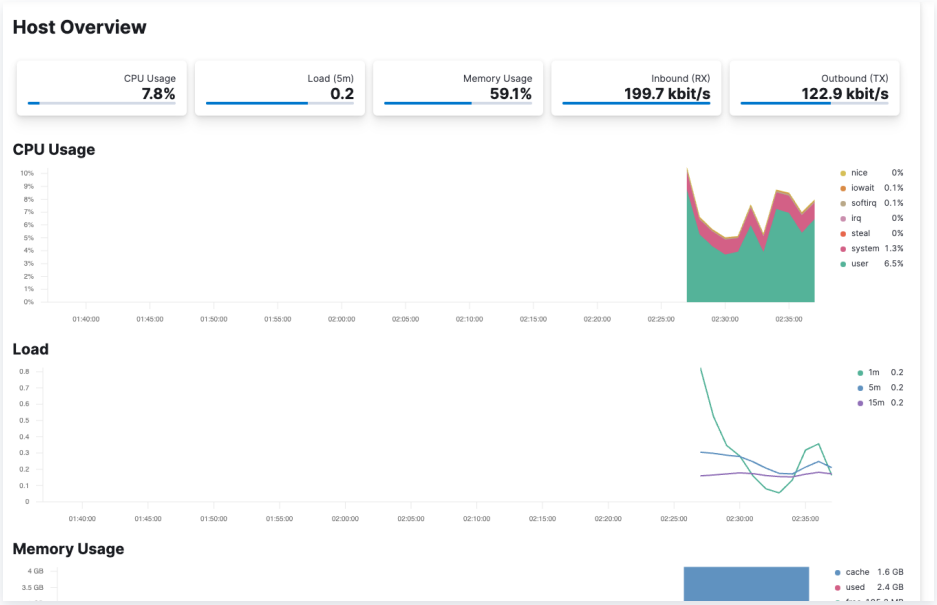


Don't forget to set up your out-of-the-box rules!





Metricbeat is also configured for monitoring the container's host through `/var/run/docker.sock` Checking Elastic Observability allows you to see metrics coming in from your host.



Filebeat

Now that the cluster is stable and monitored with Metricbeat, let's look at Filebeat for log ingestion. Here, our Filebeat will be utilized in two different ways:

docker-compose.yml ('filebeat01' container)

```
filebeat01:
  depends_on:
    es01:
      condition: service_healthy
  image: docker.elastic.co/beats/filebeat:${S
  user: root
  volumes:
    - certs:/usr/share/filebeat/certs
    - filebeatdata01:/usr/share/filebeat/data
    - "./filebeat_ingest_data/:/usr/share/file
    - "./filebeat.yml:/usr/share/filebeat/file
    - "/var/lib/docker/containers:/var/lib/doc
```

Copy

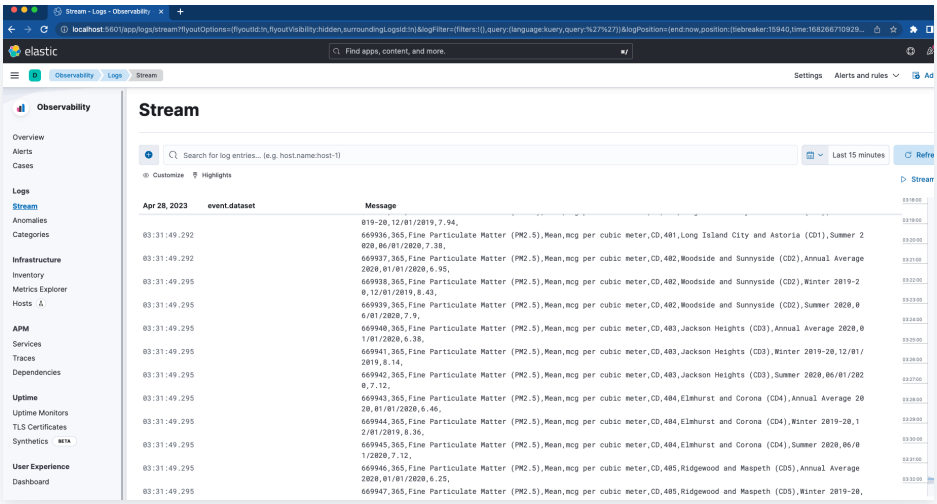
filebeat.yml

```
filebeat.inputs:
- type: filestream
 id: default-filestream
 paths:
   - ingest_data/*.log


filebeat.autodiscover:
 providers:
   - type: docker
     hints.enabled: true
```

Copy

First, we set a bind mount to map the folder “filebeat_ingest_data” into the container. If this folder doesn't exist on your host, it will be created when the container spins up. If you’d like to test the **Logs Stream** viewer within Elastic Observability for your custom logs, you can easily drop any file with a `.log` extension into `/filebeat_ingest_data/` and the logs will be read into the default Filebeat Datastream.

Alongside this, we also map in `/var/lib/docker/containers` and `/var/run/docker.sock` which, combined with the `filebeat.autodiscover` section and **hints-based autodiscover**, allows Filebeat to pull in the logs for all the containers. These logs will also be found in the Logs Stream viewer mentioned above.



Logstash

Our final container to bring to life is none other than Logstash.

docker-compose.yml (‘logstash01’container)

```
  logstash01:
    depends_on:
      es01:
        condition: service_healthy
      kibana:
        condition: service_healthy
    image: docker.elastic.co/logstash/logstash:$
    labels:
      co.elastic.logs/module: logstash
    user: root
    volumes:
      - certs:/usr/share/logstash/certs
```

Copy

logstash.conf

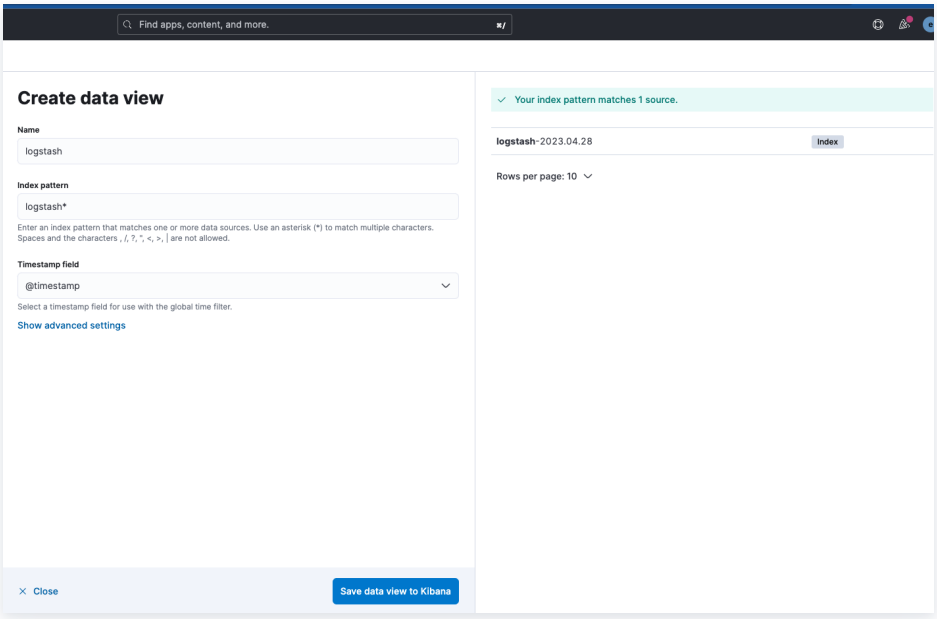
```
input {
 file {
```

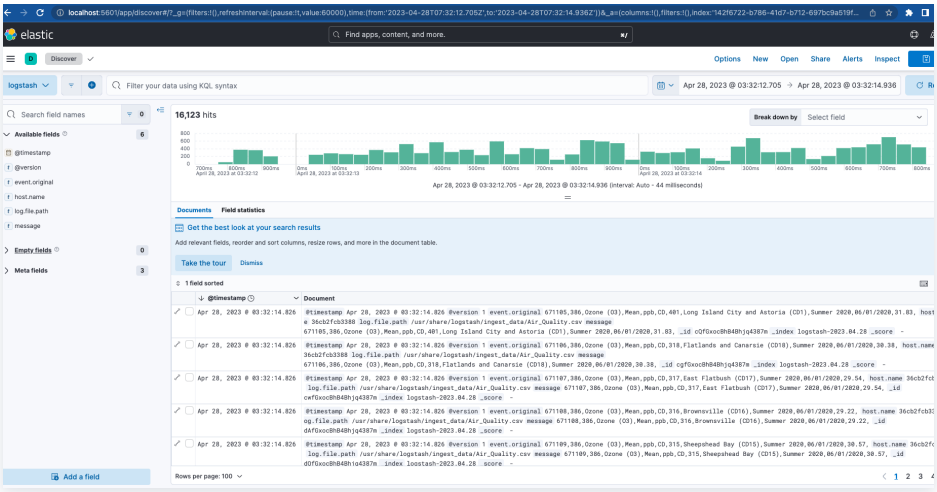
```
    #https://www.elastic.co/guide/en/logstash/cu
    #default is TAIL which assumes more data wi
    #change to mode => "read" if the file is a c
    mode => "tail"
    path => "/usr/share/logstash/ingest_data/*"
  }
}


filter {
```

Copy

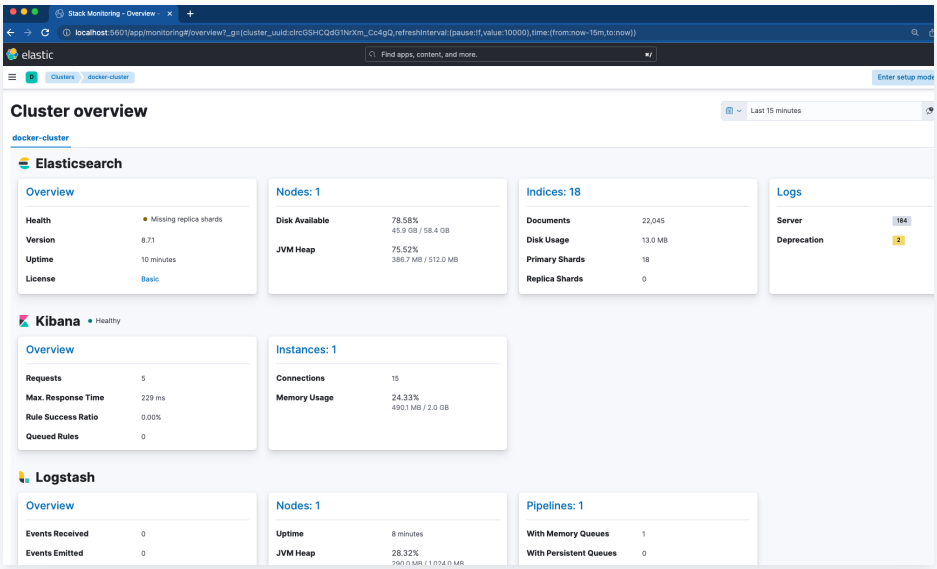
configuration. Again we're using a bind mount and mapping a folder called `/logstash_ingest_data/` from the host into the Logstash container. Here, you can test out some of the many **input plugins** and **filter plugins** by modifying the logstash.yml file. Then drop your data into the `/logstash_ingest_data/` folder. You may need to restart your Logstash container after modifying the logstash.yml file.

Note, the Logstash output index name is "logstash-%{+YYYY.MM.dd}". To see the data, you will **need to create a Data View** for the "logstash-*" pattern, as seen below.

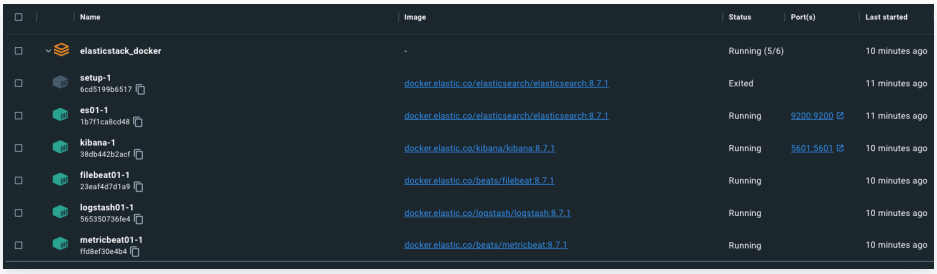




Now, with Filebeat and Logstash both up and running, if you navigate back to Cluster Monitoring you will see Logstash being monitored, as well as some metrics and links for Elasticsearch Logs.



Conclusion



Part one of this series has covered a full active cluster with monitoring and ingestion as the foundation of our stack. This will

act as your local playground to test some of the features of the Elastic ecosystem.

Ready to learn more? **Check out part two**! We dive into optimizing this foundation, along with setting up additional features such as APM Server, Elastic Agents, Elastic Integrations, and Elastic Search. We also deploy and test an application that you can instrument with some of these pieces.

All files discussed here **are available on GitHub** along with some sample data to ingest for Filebeat and Logstash.

Watch the introduction to Elastic Stack

Additional resources

1. **Running Elasticsearch on Docker**
2. **Running Kibana on Docker**
3. **Running Metricbeat on Docker**
4. **Running Filebeat on Docker**
5. **Running Logstash on Docker**