How to integrate Elasticsearch with Connectware

by Klaus Pittig & Jan Erichsen

Prerequisites

In this lesson, we will send data from Cybus Connectware to an Elasticsearch Cluster.

As a prerequisite, it is necessary to set up the Connectware instance and the Elasticsearch instance to be connected. In case of joining a more advanced search infrastructure, a Logstash instance between the Connectware and the Elasticsearch cluster may be useful.

We assume you are already familiar with Cybus Connectware and its service concept. If not, we recommend reading the articles Connectware Technical Overview and Service Basics for a quick introduction. Furthermore, this lesson requires basic understanding of MQTT and how to publish data on an MQTT topic. If you want to refresh your MQTT knowledge, we recommend looking at the lessons MQTT Basics and How to connect an MQTT client to publish and subscribe data.

Introduction

This article provides general information about Elasticsearch and its role in the Industrial IoT context along with a hands-on section about the Cybus Connectware integration with Elasticsearch.

Feel free to skip the first part if you are familiar with Elasticsearch and its ecosystem, and jump directly to the hands-on section: Using Filebeat Docker containers with Cybus Connectware.

The article concludes by describing some aspects of working with relevant use cases for prototyping, design decisions and reviewing the integration scenario.

Elasticsearch

Elasticsearch is an open-source enterprise-grade distributed search and analytics engine built on Apache Lucene. Lucene is a high-performance, full-featured text search engine programming library written in Java. Since its first release in 2010, Elasticsearch has become widely used for full-text search, log analytics, business analytics and other use cases.

Elasticsearch has several advantages over classic databases:

- It can be used to search for all kinds of documents using modern search engine capabilities.

- It supports a JSON-based domain-specific language (DSL), a RESTful API, TLS for encrypted communication, multitenancy and role-based access control for cluster APIs and indices.
- It is highly scalable and supports near real-time search.
- It is distributed, which means that indices can be divided into shards with zero or more replicas while routing and rebalancing operations are performed automatically.
- It supports real-time GET requests, which makes it suitable as a NoSQL datastore.

Mastering these features is a known challenge, since working with Elasticsearch and search indices in general can become quite complex depending on the use case. The operational effort is also higher, but this can be mitigated by using managed Elasticsearch clusters offered by different cloud providers.

Elasticsearch inherently comes with a log aggregator engine called Logstash, a visualization and analytics platform called Kibana and a collection of data shippers called Beats. These four products are part of the integrated solution called the “Elastic Stack”. Please follow the links above to learn more about it.

Industrial IoT with the Elastic Stack

When it comes to Industrial IoT, we speak about collecting, enriching, normalizing and analyzing huge amounts of data ingested at a high rate even in smaller companies. This data is used to gain insights into production processes and optimize them, to build better products, to perform monitoring in real time, and last but not least, to establish predictive maintenance. To benefit from this data, it needs to be stored and analyzed efficiently, so that queries on that data can be made in near real time.

Here a couple of challenges may arise. One of them could be the mismatch between modern data strategies and legacy devices that need to be integrated into an analytics platform. Another challenge might be the need to obtain a complete picture of the production site, so that many different devices and technologies can be covered by an appropriate data aggregation solution.

Some typical IIoT use cases with the Elastic Stack include:

- predictive analytics leading to predictive maintenance (which allows better maintenance plans, cost savings and less outages in the production process)
- reducing reject rate in the production process
- ensuring shop floor security
- real time machine status monitoring

Elastic Stack has become one of several solutions for realizing such use cases in a time and cost efficient manner, and the good thing is that Elasticsearch can be easily integrated into the shop floor with Cybus Connectware.

Cybus Connectware is an ideal choice for this because its protocol abstraction layer is not only southbound protocol agnostic, supporting complex shop floor environments with many different machines, but also agnostic to the northbound systems. This means that customers can remain flexible and realize various use cases according to their specific requirements. For example, migrating a Grafana-based local monitoring system to Kibana, an Elastic Stack-based real time monitoring Dashboard, is a matter of just a few changes.

Connectware & Elasticsearch Integration

The learning curve for mastering Elasticsearch is steep for users who try it for the first time. Maintaining search indices and log shippers can also be complex for some use cases. In those cases, it might be easier and more efficient to integrate machines into an IIoT edge solution, such as Connectware.

Here are some benefits of using Cybus Connectware at the edge to stream data to Elasticsearch:

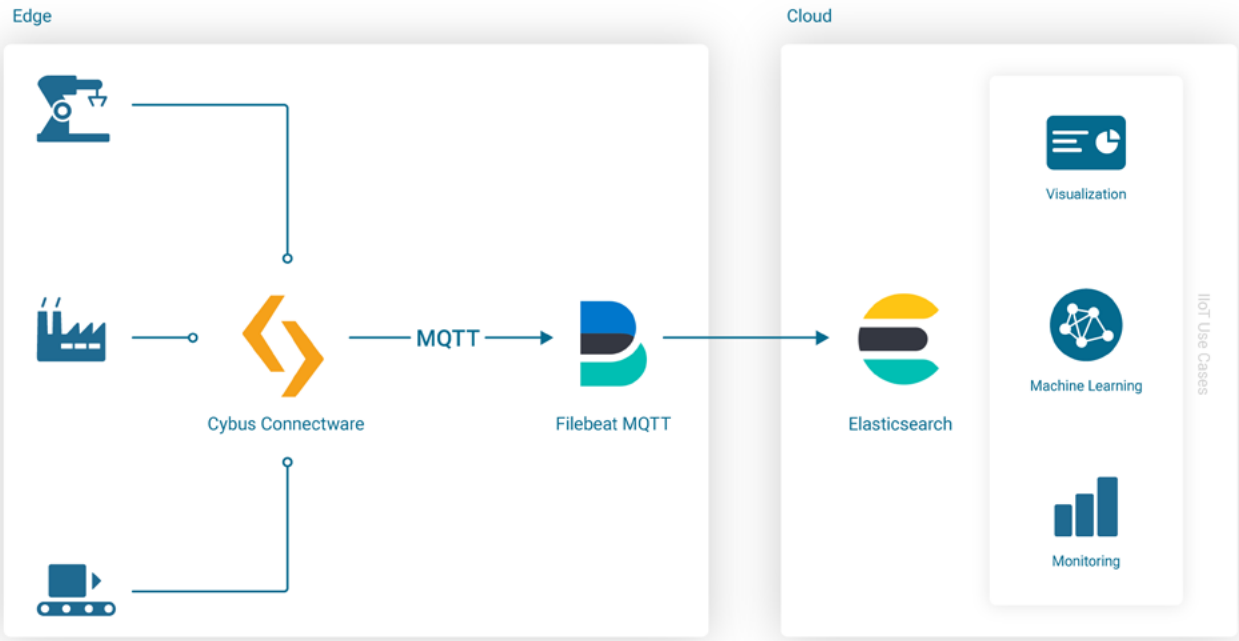
- Complex machine data is made available in a standardized format for use in Elasticsearch
- As an OT/IT gateway, Connectware allows full control over all data transmitted from machines to the northbound systems, which covers important aspects of data governance
- Changes in specified data formats can easily be covered by using a low-code approach for mappings and transformation rules in Connectware service commissioning files
- In case of connectivity issues to endpoints like an Elasticsearch cluster, machine data is cached and transmitted without loss after the connection was restored
- Different targets for machine data can be added at any time to existing services, so that smooth data migration to different and multiple northbound endpoint targets is possible

Nevertheless, when ingesting data the Filebeat and Logstash data processing features may also be very useful for normalizing data for all data sources, not just IIoT data from OT networks.

Before proceeding further, you should first obtain access credentials for an existing Elasticsearch cluster, or set up a new one. For this, follow the instructions below:

- Getting Started with Elasticsearch

The simplest and most reliable way of communication when integrating Cybus Connectware with Elasticsearch is the MQTT Input for a Filebeat instance. An additional advantage of the Connectware MQTT connector is built-in data buffering, which means that data is stored locally when there is temporary connection failure between the Filebeat and the Elasticsearch Cluster:



Using Filebeat Docker containers with Cybus Connectware

Embedding the Filebeat Docker Image into Connectware is easy because Connectware comes with an integrated interface for running Edge Applications. Once started, the docker container connects to the integrated Connectware Broker to fetch and process the data of interest.

All you have to do is to create a Connectware Service by writing a Commissioning File and install it on the Connectware instance. To learn more about writing Commissioning Files and Services, head over to the Learn Article called Service Basics.

Now let's get straight to the point and start writing the Commissioning File.

Step 1. Create a basic template file called filebeat.yml

This is the basic structure of a commissioning file:

```
---
description: Elastic Filebeat reading MQTT Input

metadata:
  name: Filebeat

parameters:

definitions:
```

```
resources:
```

Step 2. Add Container resource for the Filebeat

Add a `Cybus::Container Resource` for the filebeat to the `resources` section in the template. This will later allow you to run the Container when installing and enabling the Service, using the docker image from docker.elastic.co directly:

```
resources:
  filebeat:
    type: Cybus::Container
    properties:
      image: docker.elastic.co/beats/filebeat:7.13.2
      volumes:
        - /var/run/docker.sock:/var/run/docker.sock:ro
```

Step 3. Specify the Filebeat configuration

When starting the filebeat container, various variables must be configured correctly. In this example, these variables should not be integrated into a specialized container image. Instead, the variables should be configured “on the fly” when starting the standard container image from within Connectware, so that the entire configuration is stored in a single commissioning file. For this purpose, all configuration settings of the filebeat container are specified in the helper section of the commissioning file, defining a variable called `CONFIG` inside of the file’s `definitions` section:

```
definitions:
  CONFIG: !sub |
    filebeat.config:
      modules:
        path: /usr/share/filebeat/modules.d/*.yml
        reload.enabled: false

    filebeat.inputs:
    - type: mqtt
      hosts:
        - tcp://${Cybus::MqttHost}:${Cybus::MqttPort}
      username: admin
```

```
      password: admin
      client_id: ${Cybus::ServiceId}-filebeat
      qos: 0
      topics:
        - some/topic

    setup.ilm.enabled: false
    setup.template.name: "some_template"
    setup.template.pattern: "my-pattern-*"

    output.elasticsearch:
      index: "idx-%{+yyyy.MM.dd}-00001"

    cloud.id: "elastic:d2V***"
    cloud.auth: "ingest:Xbc***"
```

Step 4. Tweak the Filebeat container behavior on startup

Now that the Filebeat configuration is set up, the container resource `filebeat` mentioned above needs to be extended in order to use this configuration on startup (in this and the following examples, the top-level headline `resources:` is skipped for brevity):

```
filebeat:
  type: Cybus::Container
  properties:
    image: docker.elastic.co/beats/filebeat:7.13.2
    entrypoint: [""]
    command:
      - "/bin/bash"
      - "-c"
      - !sub 'echo "${CONFIG}" > /tmp/filebeat.docker.yml && /usr/bin/tini -s
-- /usr/local/bin/docker-entrypoint -c /tmp/filebeat.docker.yml -environment
container'
    volumes:
      - /var/run/docker.sock:/var/run/docker.sock:ro
```

Step 5. Introduce parameters for access credentials

The filebeat container needs access credentials to set up the cloud connection correctly. Those credentials should not be written into the file as hard-coded values though. Credentials as hard-coded values should be avoided not only for security reasons, but also to make the commissioning file re-usable and re-configurable for many more service operators. To easily achieve this, we are going to use `parameters`.

In the `parameters` section we are creating two parameters of type string:

```
parameters:
  filebeat-cloud-id:
    type: string
    description: The cloud id string, for example elastic:d2V***
 filebeat-cloud-auth:
    type: string
    description: The cloud auth string, for example ingest:Xbc***
```

These `parameters` are now ready to be used in our configuration. During the installation of the service, Connectware will ask us to provide the required values for these `parameters`.

To use the parameters in the configuration, the following lines in the Filebeat configuration (the `CONFIG` definition from above) need to be adapted:

```
  cloud.id: "${filebeat-cloud-id}"
  cloud.auth: "${filebeat-cloud-auth}"
```

Step 6. Replace broker credentials with Connectware parameters

The filebeat container is using access credentials not only for the cloud connection but also for the local input connection, which is the connection to the Connectware MQTT broker. Those access credentials have been set to the default credentials (admin/admin) in the definition above, which now need to be adapted to the actual non-default credentials. For your convenience, Connectware already has Global Parameters that are replaced by the current credentials of the MQTT broker. So the following lines in the Filebeat configuration (the `CONFIG` definition from above) need to be adapted, too:

```
  username: ${Cybus::MqttUser}
  password: ${Cybus::MqttPassword}
```

Step 7. Configure read permissions for topics

Finally the `defaultRole` for this service requires additional read permissions for all MQTT topics which the service should consume. To grant these additional privileges, another resource should be added:

```
resources:
  defaultRole:
    type: Cybus::Role
    properties:
      permissions:
        - resource: some/topic
          operation: read
          context: mqtt
```

Final service commissioning file

In the end, the entire service commissioning file should look like this:

```
---
description: Elastic Filebeat reading MQTT Input

metadata:
  name: Filebeat

parameters:
  filebeat-cloud-id:
    type: string
    description: The cloud id string, for example elastic:d2V***

  filebeat-cloud-auth:
    type: string
    description: The cloud auth string, for example ingest:Xbc***

definitions:
  # Filebeat configuration
  CONFIG: !sub |
    filebeat.config:
      modules:
        path: /usr/share/filebeat/modules.d/*.yml
```

```
        reload.enabled: false

    filebeat.inputs:
    - type: mqtt
      hosts:
        - tcp://${Cybus::MqttHost}:${Cybus::MqttPort}
      username: ${Cybus::MqttUser}
      password: ${Cybus::MqttPassword}
      client_id: ${Cybus::ServiceId}-filebeat
      qos: 0
      topics:
        - some/topic

    setup.ilm.enabled: false
    setup.template.name: "some_template"
    setup.template.pattern: "my-pattern-*"

    output.elasticsearch:
      index: "idx-%{+yyyy.MM.dd}-00001"

    cloud.id: "${filebeat-cloud-id}"
    cloud.auth: "${filebeat-cloud-auth}"

resources:
  # The filebeat docker container
  filebeat:
    type: Cybus::Container
    properties:
      image: docker.elastic.co/beats/filebeat:7.13.1
      entrypoint: [""]
      command:
        - "/bin/bash"
        - "-c"
        - !sub 'echo "${CONFIG}" > /tmp/filebeat.docker.yml && /usr/bin/tini
-- /usr/local/bin/docker-entrypoint -c /tmp/filebeat.docker.yml -environment
container'
      volumes:
        - /var/run/docker.sock:/var/run/docker.sock:ro
```

```
  # Gaining privileges
  defaultRole:
    type: Cybus::Role
    properties:
      permissions:
        - resource: some/topic
          operation: read
          context: mqtt
```

This commissioning file can now be installed and enabled, which will also start the filebeat container and set up its connections correctly. However, there is probably no input data available yet, but we will get back to this later. Depending on the input data, an additional structure should be prepared for useful content in Elasticsearch, which is described in the next section.

Improving attributes for more suitable content structure in Elasticsearch

The first contact with the Elasticsearch cluster can be verified by sending some message to the topic to which the Filebeat MQTT inputs are subscribed (here: “some/topic”) and reviewing the resulting event in Kibana.

Once this is done, a service integrator may identify several elements of the created JSON document that need to be changed. The deployed Connectware service commissioning file allows us to ship incoming MQTT messages in configured topics to the Elasticsearch cluster as a JSON document with certain meta data that an operator may want to change to improve the data source information.

For example, a message sent using the service described above contains multiple fields with identical values, in this case agent.name, agent.hostname and host.name. This is due to the naming convention for container resources in a service commissioning files described in the Connectware Container Resource. As the ServiceId is “filebeat”, and the container resource is named “filebeat” too, the resulting container name, hostname and agent name in the transmitted search index documents are “filebeat-filebeat”, which looks as follows:

```
...
"fields": {
...
   "agent.name": [

     "filebeat-filebeat"
```

```
    ],
    "host.name": [
      "filebeat-filebeat"
    ],
...
    "agent.hostname": [
      "filebeat-filebeat"
    ],
...
```

To get appropriate names in the search index for further evaluation and post processing, either change the serviceId and/or container resource name in the service commissioning file, or use Filebeat configuration options to set an alternative agent.name (by default is derived from the hostname, which is the container hostname created by Connectware). Be aware that the maximum number of characters for the clientId in Filebeat mqtt.input configuration is 23.

Example

Change both the service name (serviceId) and the container resource name to identify the respective device as the data source, and redeploy the service commissioning file:

```
...
metadata:
  name: Shopfloor 1

...
resources:
  # The filebeat docker container
  filebeat_Machine_A_032:
...
```

In addition to this, the Filebeat configuration can be modified slightly to set the agent.name appropriately along with some additional tags to identify our edge data sources and data shipper instance (is useful to group transactions sent by this single Beat):

```
...
definitions:
  CONFIG: !sub
```

```
...
   name: "shopfloor-1-mqttbeat"
   tags: [ "Cybus Connectware", "edge platform", "mqtt" ]
...
```

This leads to improved field values in the search index, so that transactions can be better grouped in the search index, such as this:

```
...
"fields": {
...
   "agent.name": [
     "shopfloor-1-mqttbeat"
   ],
   "host.name": [
     "shopfloor-1-mqttbeat"
   ],
...
   "agent.hostname": [
     "shopfloor1-filebeat_Machine_A_032"
   ],
...
   "tags": [
     "Cybus Connectware",
     "edge platform",
     "mqtt"
   ],
...
```

Providing machine data for Elasticsearch

Using Cybus Connectware offers extensive flexibility in mapping devices, configuring pre-processing rules and adding many different resources. It is up to the customer to define the requirements, so that a well-architected set of services can be derived for the Connectware instance.

To stream machine data collected by Connectware to Elasticsearch, existing MQTT topics can be subscribed by the Filebeat container. Alternatively, the Filebeat container can subscribe to MQTT topics that contain

specific payload transformation. For instance, a normalized payload for an Elasticsearch index specifies an additional timestamp or specific data formats.

The advantage of using Connectware to transmit data to Elasticsearch is that it supports a lightweight rules engine to map data from different machines to Filebeat by just working with MQTT topics, for example:

```
resources:
  # mapping with enricher pattern for an additional timestamp
  machineDataMapping:
    type: Cybus::Mapping
    properties:
      mappings:
       - subscribe:
            topic: !sub '${Cybus::MqttRoot}/machineData/+field'
          rules:
            - transform:
                expression: |
                  (
                    $d := { $context.vars.field: value };
                    $merge(
                      [
                        $last(),
                        {
                          "coolantLevel": $d.`coolantLevel`,
                          "power-level": $d.`power-level`,
                          "spindleSpeed": $d.`spindleSpeed`,
                          "timestamp": timestamp
                        }
                      ]
                    )
                  )
            publish:
              topic: 'some/topic'
```

A reasonable structural design of related Connectware service commissioning files depends on the number of machines to connect, their payload, complexity of transformation and the search index specifications in the Elasticsearch environment. See the Github project for a more advanced example concerning machine data transformation.

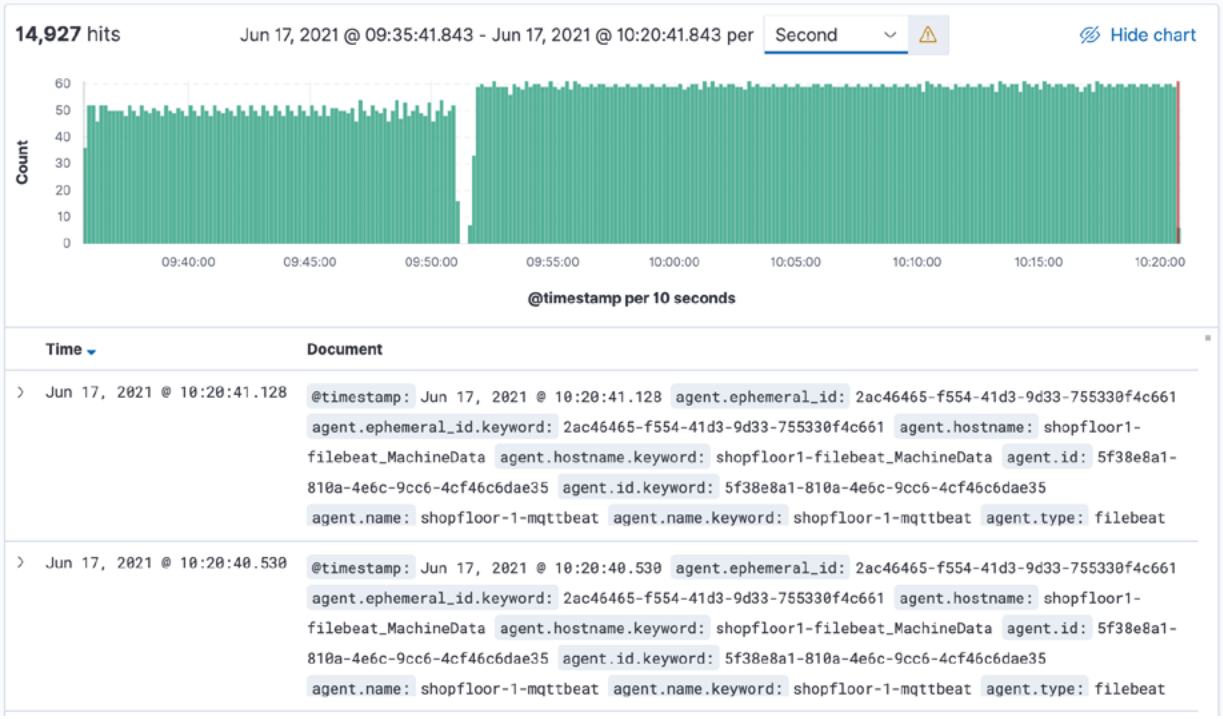
To explain these settings in detail, Cybus provides a complete Connectware documentation and Learn articles like Service Basics.

Discover, visualize and analyze data in Elasticsearch

What has been added to the original Filebeat configuration is the typical task of a service operator connecting shopfloor devices and organizing respective resources in a Connectware service commissioning file. The service operator has further options to decompose this file to multiple files to optimize the deployment structure in this Low-code/No-code environment for their needs. Contact Cybus to learn more about good practices here.

Now that the data is transmitted to the Elasticsearch cluster, further processing is up to the search index users. The Elastic Stack ecosystem provides tools for working with search indices created from our data, such as simple full text search with Discovery, Kibana visualizations or anomaly detection and so on.

The message is transmitted as a message string and will be stored as a JSON document with automatically de-composed payload and associated metadata for the search index. A simple discovery of that continuously collected data shows something like this:



Where to go from here

This lesson offered a brief introduction into the integration of Cybus Connectware with Elasticsearch using the Connectware built-in MQTT connector and the FileBeat with MQTT Input in a service commissioning file.

Cybus also provides sample service commissioning files and some other technical details in the Github project How to integrate Elasticsearch with Connectware.

As a next step, you can use Cybus Connectware to organize data ingestion from multiple machines for its further use with the Elastic Stack.

Cybus is a specialist for secure IIoT Edge software, headquartered in Germany. Cybus Connectware serves smart factories as a universal Edge and DevOps hub. Machine builders and providers of IIoT services use the Cybus Connectware as a software-based gateway. As early as 2017, Cybus published the first secure industrial connector for machine data according to today's DIN SPEC 27070 standard. Industry analyst Gartner named Cybus a worldwide "Cool Vendor". Today, the company counts medium-sized and large companies from numerous industrial sectors such as mechanical engineering, automotive and aviation among its customers.

Cybus GmbH · Osterstraße 124 · 20255 Hamburg · Germany · www.cybus.io · hello@cybus.io · (+49) 40 228 58 68 50