**Hands On Create Logstash Pipeline**

Creating a Logstash pipeline

Creating a Logstash pipeline involves defining a series of configurations that specify how data should be collected, parsed, filtered, and where it should be sent for further analysis or storage. This process enables you to harmonize data from various sources, making it ready for visualization, search, and analysis through Elasticsearch and Kibana, as well as third-party destinations.

In this recipe, we will walk you through the steps to create a Logstash pipeline, from configuring input sources to defining filters, specifying output destinations, and running the pipeline. Our example is based on Rennes Traffic Data, which we introduced in [*Chapter 4*](https://subscription.packtpub.com/book/data/9781837634293/4).

Getting ready

You will need to have completed the previous *Installing self-managed Logstash* recipe and the *Setting up time series data stream (TSDS) manually* recipe in [*Chapter 4*](https://subscription.packtpub.com/book/data/9781837634293/4) as we are going to reuse the objects that we created in this recipe such as the index life cycle policy, mapping, setting, and the index template.

The dataset we will use is the one introduced in [*Chapter 4*](https://subscription.packtpub.com/book/data/9781837634293/4), Rennes Traffic Data. This time, however, instead of importing the latest traffic status of Rennes just once, we will create a new data stream to continuously receive real-time traffic data. This means that every 10 minutes, we will import the latest traffic data from the public data API and append it to the data stream.

The snippets of this recipe are available at <https://github.com/PacktPublishing/Elastic-Stack-8.x-Cookbook/blob/main/Chapter5/snippets.md#creating-a-logstash-pipeline>.

How to do it...

You will need to first create the Logstash configuration file in the Logstash configuration folder (/etc/logstash/conf.d/ for Debian). The reference configuration file can be found at this address: <https://github.com/PacktPublishing/Elastic-Stack-8.x-Cookbook/blob/main/Chapter5/logstash-conf/rennes_traffic-default.conf>. If you are not deploying Logstash on Debian, the details of the Logstash directory layout for different installation packages can be found here: <https://www.elastic.co/guide/en/logstash/current/dir-layout.html>.

Now let’s go through the process of setting up a Logstash configuration to fetch and process real-time traffic data from Rennes. We’ll outline each section of the configuration file, beginning with the input plugin and concluding with the output configuration. By following these instructions, you’ll be able to ingest and later visualize traffic data.

1. Let us start by configuring the input plugin, which specifies where Logstash should fetch data from. Here, we are using the http\_poller input plugin to fetch the Rennes traffic data every 10 minutes in CSV format:

input {

    http\_poller {

        urls => {

            rennes\_data\_url => "https://data.rennesmetropole.fr/explore/dataset/etat-du-trafic-en-temps-reel/download?format=csv&timezone=Europe/Paris&use\_labels\_for\_header=false"

        }

        request\_timeout => 60

        schedule => { every => "10m" }

        codec => "line"

    }

}

1. Next, let us prepare the filter section to format the data correctly before ingesting it into our data stream. We are using different filters in this order:
   1. The csv filter is used to parse the data from different columns and remove unwanted fields.
   2. Fetch and parse the datetime field and mutate it to index the @timestamp field.
   3. Convert the Sens unique column to a Boolean field that we name oneway.
   4. Rename the fields according to our field mapping for the data stream.
   5. Remove the unnecessary fields that have been generated from the csv filter:

filter{

    csv {

        separator => ";"

        skip\_header => "true"

        columns => [

            "datetime","predefinedlocationreference",

            "averagevehiclespeed","traveltime",

            "traveltimereliability","trafficstatus",

            "vehicleprobemeasurement","geo\_point\_2d",

            "geo\_shape","gml\_id","id\_rva\_troncon\_fcd\_v1\_1",

            "hierarchie","hierarchie\_dv","denomination",

            "insee","sens\_circule","vitesse\_maxi"

        ]

        remove\_field => [

            "geo\_shape","gml\_id",

            "id\_rva\_troncon\_fcd\_v1\_1"

        ]

    }

    date {

        match => ["datetime", "UNIX"]

        target => "@timestamp"

    }

    if [sens\_circule] == "Sens unique" {

        mutate {

            add\_field => { "oneway" => "true" }

        }

    }

    else {

        mutate {

            add\_field => { "oneway" => "false" }

        }

    }

    mutate {

        rename => {"traveltime" => "traveltime.duration"}

        ...

    }

    mutate {

        remove\_field => [

            "datetime","message","path","host","@version",

            "original","event.original","tags","sens\_circule"

        ]

    }

}

1. Now, let’s finish preparing the Logstash configuration file. We will focus on the **output configuration**. Here, we will set two outputs, one to Elastic Cloud with specific settings for a metrics data stream and the second for STDOUT to display the output for debugging purposes in the shell running Logstash:

output {

    elasticsearch {

        cloud\_id => "CLOUD\_ID"

        cloud\_auth => "user:password"

        data\_stream => true

        data\_stream\_type => "metrics"

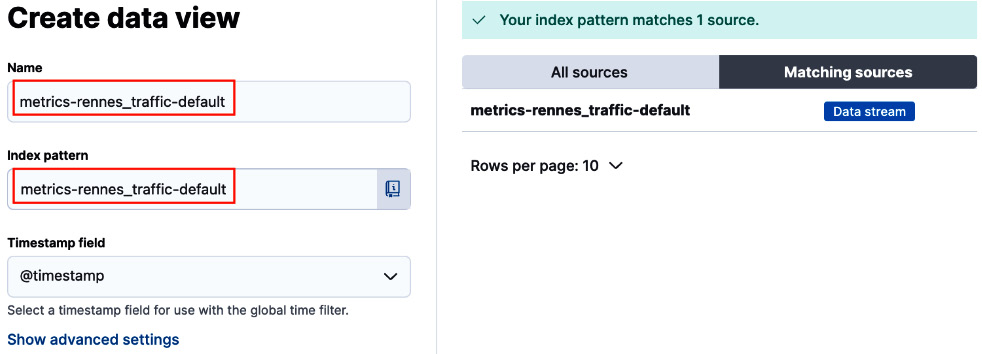
        data\_stream\_dataset => "rennes\_traffic"

        data\_stream\_namespace => "default"

    }

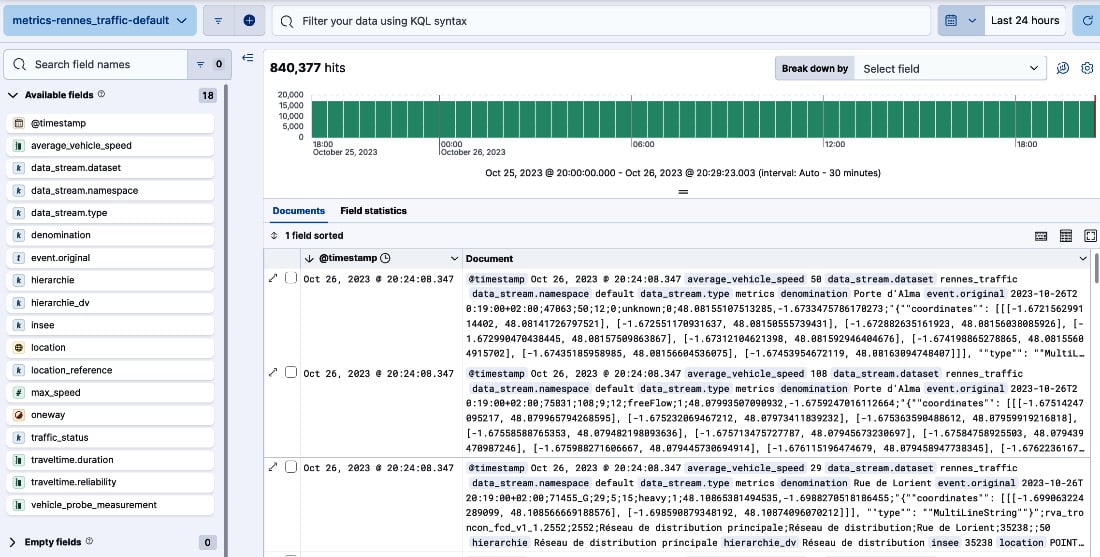
    stdout { codec => rubydebug }

}

1. Once we finish the configuration file, we can start Logstash with the following command (for Debian):  
     
     
     
   We can now check the data ingestion by creating a data view in Kibana. Go to Kibana, **Stack Management** | **Data Views** | **Create Data View**, and then set the name and index pattern with metrics-rennes\_traffic-default:  
     
   

$ sudo systemctl start logstash.service

1. We can then go to **Kibana** | **Analytics** | **Discover** in Kibana and choose metrics-rennes\_traffic-default to verify that the data has been successfully ingested, as shown in *Figure 5.15*:



How it works...

As you can see in *Figure 5.16*, each Logstash instance can host multiple pipelines:

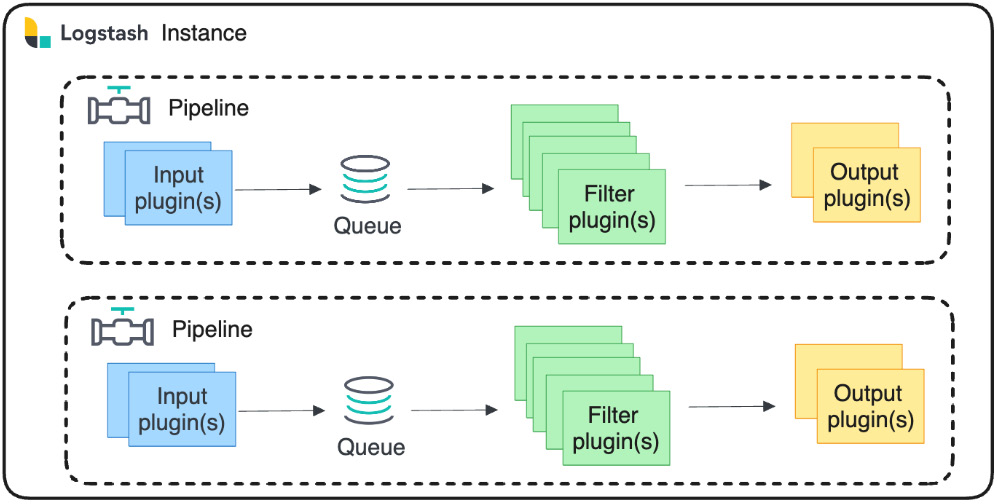
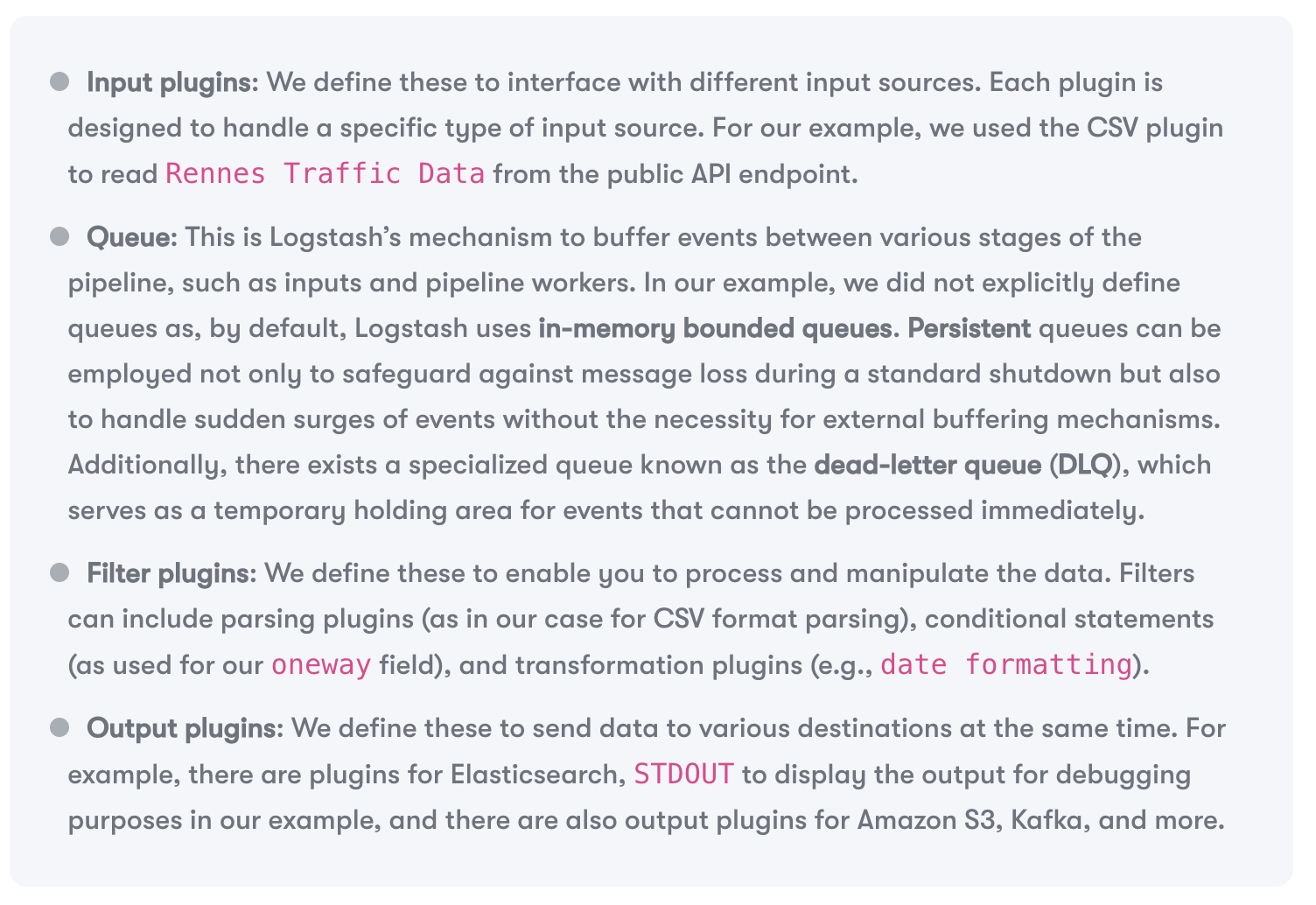


Figure 5.16 – Logstash pipelines

In each pipeline, we define the following:



## There’s more...

As mentioned earlier in this chapter, there are two possible ways to create data transformation pipelines: using ingest pipelines or Logstash pipelines. Both have pros and cons. Here is a comparison of both methods:

