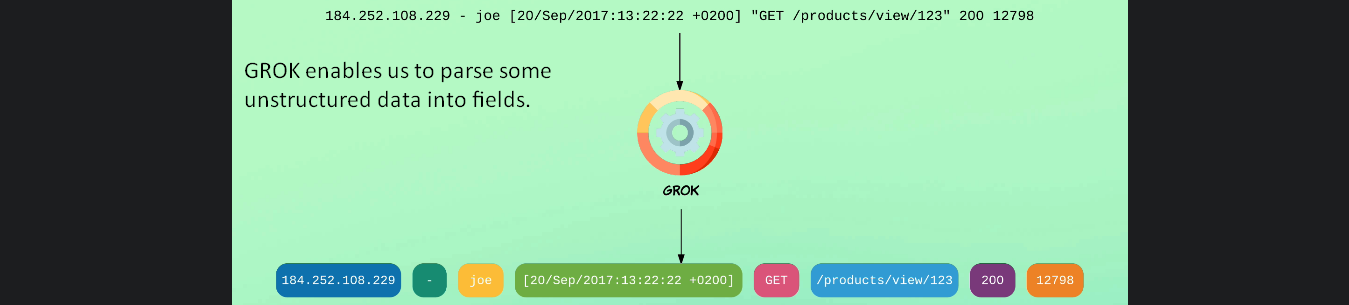
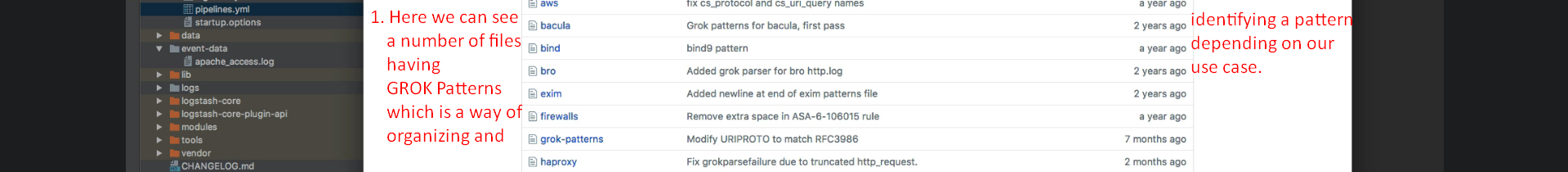
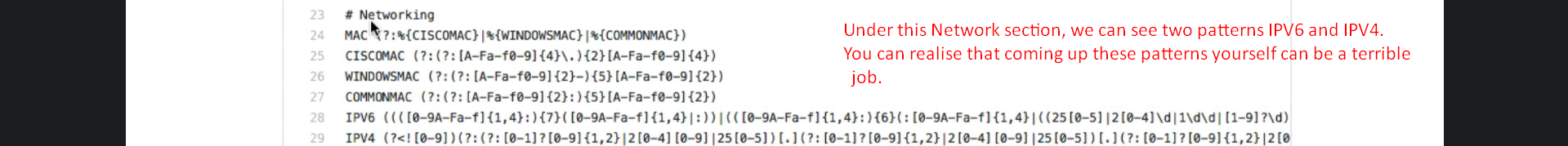
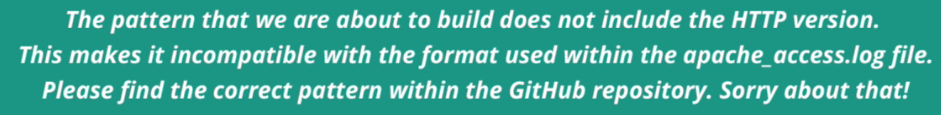
1. 
2. In the lecture, we did basic setup, Let’s do something useful with the **Access Log**.
3. **Agenda**:
   1. To parse the incoming events.
      1. Right now, the **event data** is just put into the field name “message” but we want to split the **event data** into **multiple fields**.
      2. **For Example**:
         1. We want the HTTP Status to be a separate field,
         2. the IP to be another, and
         3. the HTTP Verb to be another one.
      3. **Benefits**:
         1. Doing this enables us to do some processing with the values which we will see later.
         2. This also means we can output something that is meaningful to the system that receives the output.
            1. **For example**:  
               Example could be to send JSON off to an HTTP Endpoint.  
               or to index the results as a document in **Elasticsearch**.
4. How to parse the message field and create new fields from the certain parts (not necessary to parse all the values from a message) of the message field’s value?
5. We can use that using a **Filter Plugin** called 🡺 **GROK**  
   which is **intended for making sense of unstructured Data**.
6. **GROK enables us to parse some unstructured data into fields.**
7. We can do that by specifying **text patterns**, which you can think of as variables containing **regular expressions**.  
   We can type our own regular expressions but as GROK ships with lots of built-in patterns. So, you will rarely need to do this.   
   
8. The following is the **format pattern syntax**.
   1. Which starts with % symbol.
   2. Within the curly brackets, we configure the pattern.
   3. **SYNTAX**: **Name of the pattern** that will be used to match some text.
      1. Here we can define any of **Grok’s predefined patterns**.
      2. Very conveniently, there is Github link having all those patterns related to any technologies.  
         <https://github.com/logstash-plugins/logstash-patterns-core>
   4. The following patterns folder contains the GROK patterns.   
        
        
        
        
      Here you can see files having patterns where each file contains patterns specific to a particular technology.
9. We will come back to more specific GROK Pattern in a moment.  
   First, let’s see the basic patterns not related to any specific technology.  
   These patterns can be found inside the grok-patterns text file.
10. 
11. In this file, we can see a number of patterns.
12. Each pattern consists of **Identifier = name** and a regular expression.
13. There are a number of patterns from numbers to words, email addresses, dates to URLs.
14.   
    So, just refer to these patterns identifiers and we don’t have to deal with writing the regular expressions ourselves and handling a world of edge cases.
15. So, what we do when specifying the pattern identifier, is that we are defining the format that a piece of text should be in for our pattern to match.  
    **For example:** We can use the email address pattern to specify that a given piece of text should be an email address.
16. Okay, so this is the syntax of the **GROK Syntax**.  
    A picture containing text, font, screenshot, graphics

    Description automatically generated
17. Let’s go through a couple of an examples.
18. Suppose we have a String of text containing a name of a person, email address, and age.  
    A blue rectangle with white text

    Description automatically generated with medium confidence
19. Now we want to parse this unstructured text and store the values into some fields beginning with the first name.  
    To match this, we can use the predefined pattern, named word for the **SYNTAX** path **%{SYNTAX : SEMENTIC}**
20. A picture containing text, screenshot, font, diagram

    Description automatically generated  
    Now we know the very basics of the GROK Pattern.  
    Let’s put our knowledge to use by adding a **GROK Filter**.
21. Let’s go to the pipeline configuration file.
22. The following is the sample Apache Access Log msg which we want to parse.  
    
23. A screen shot of a computer

    Description automatically generated with medium confidence
    1. So, this match option is where we will be configuring how a filter will do its matching.
    2. Here we have to add the key with the name of the field that we want to perform the matching on then specify the **GROK Pattern** as the value.
    3. In this case, we want to match the message field of the processed event as this is the field that contains the raw event.
    4. **Let’s start specifying the patterns to parse the Apache access log file**.
    5. 
    6. **The first part of the raw event** is the IP Address of the Visitor.  
       Let’s use IP Pattern (Here IP is actually pattern identifier which is already pre-defined) which is basically a wrapper around IPV4 and IPV6.  
       **Sample**:   
       **Matching Pattern**: **🡺 %{IP:ip\_address}**
    7. Following the IP, is the identity of the visitor and then the name of the user who authenticated with HTTP basic auth.

for both of these, we can make use the pattern name USER which specifies a regular expression for username allowing them to consist of alphanumeric characters, dashes, & underscore.  
**Sample Identity of the Visitor**:   
**Matching Pattern: %{USER:identity}  
  
Sample for authenticated User:  
 Matching Pattern: %{USER:auth}**

* 1. Next part of the log statement is the timestamp.  
     GROK contains various dates and time patterns that we can use to specify a format matching the one the Apache Web Server uses but that is very verbose and inconvenient (Basically here trying to say that we should collect the blocks of time and date patterns to form the final pattern which matches the timestamp in Log). But there is a convenient pattern that can help us out with this.   
     The following GROK Patter is from the github link.  
     [](https://github.com/hpcugent/logstash-patterns/blob/master/files/grok-patterns#:~:text=facility%7D.%25%7BNONNEGINT%3Apriority%7D%3E-,HTTPDATE,-%25%7BMONTHDAY%7D/%25%7BMONTH%7D/%25%7BYEAR)  
     Let us use it.  
     **Sample**:   
     **Matching Pattern: \ [%{HTTPDATE:req\_ts}\]**
  2. Next Part of the log statement is HTTP Method and Request Path within double quotation.  
     **Sample**:   
     **Matching Pattern**: **\ “%{WORD:http\_verb} %{URIPATHPARAM:req\_path}\”**
  3. Next, we have HTTP Status Code and number of bytes:  
     **Sample**:   
     **Matching Pattern**: **%{INT:http\_status} %{INT:num\_bytes} 🡸 By default the value will be parsed to string but we want as integer.  
     %{INT:http\_status:int} %{INT:num\_bytes:int}**

1. Let’s text our configuration.
2. To make testing easier to test pipeline configuration, we will be sending HTTP request instead of removing sincedb files again and again.   
   The postman request body will contain a single entry from an access log file, i.e. a single request from Apache Access Log file.
3. A screenshot of a computer

   Description automatically generated with low confidence
4. A picture containing text, screenshot, display, design

   Description automatically generated