1. 
2. Agenda:
   1. How to prepare pipeline for handling error logs as error logs may contain multiple lines even though error log we handled already with single line.
3. Even though we have already prepared the pipeline handling the error log but still we have to do more work as error log may span multiple lines as in case of error log having stack Trace but the current pipeline is capable to handle only one line error logs.
4. **For example**:
   1. Stack trace may span multiple lines unlike access log where each line responds to a single request.
5. **Example**:  
   A screenshot of a computer

   Description automatically generated with medium confidence  
   We will work with this Java Stack Trace above where event is spanning on multiple lines.
6. But how we deal with multiline events, it depends on the structure of the events.  
   So, we need to adjust a little bit depending on our particular event like events from PHP, C#.  
   So, we need to tell Logstash which lines form an event so that it doesn’t consider each line as an individual events.
7. We can do this by using a **CODEC named Multiline**.
8. Until now, we were distinguishing b/w the two event types (access, error) within the filter block but how we are going to that in this case.
   1. There used to be a **filter plugin named Multiline** which enabled us to check the event type and conditionally apply the filter.
   2. However, this plugin is **deprecated in favour of the Multiline Codec**.
   3. But we can’t use conditional statements in input block for several reasons.
      1. First of all, type field is not set at this state of pipeline (during input stage).
      2. Secondly, we can’t apply Codec conditionally.  
         This leaves us with two options.
9. **Two options are**:
   1. Having input plugin dedicated to each event type. 🡺 We will use this approach.
   2. Or exploiting the fact that the Multiline Codec adds a tag named multiline to multi-line events.  
      It means that if an event spans multiple lines, a tag named multiline is added to the event or otherwise no tag is added.  
      This will help us to distinguish b/w multi-line events being error logs from single line events being access log.
10. Both are decent approach, but we will go with the first one.  
    It means we will dedicate an input plugin to the error logs.
11. We will add a new file input plugin that is dedicated to errors.  
    There is a reason why we are not going to use HTTP in A screen shot of a computer

    Description automatically generated put plugin as there is subtle technicality that would cause problems for us.  
    We will get back to that later in this course.  
    For now, just add file input plugin.
12. Let’s take a moment of **how a multiline codec works**.
13. Just take a look at the following snapshot:  
    A screen shot of a computer

    Description automatically generated
    1. The codec will receive one line at a time.
    2. So, the regular expression should match something that indicates that the current line is part of the multi-line event and this is not an event on its own.
    3. Let’s take a look at the Stack Trace again for a moment.  
       A screenshot of a computer

       Description automatically generated
    4. When a code receives the first line, it considers it to be an event but when the codec receives the 2nd line, we need to tell codec that this line belongs to the same event instead of a new one.
    5. Like we said, we do this by matching lines that should not trigger a new event.  
       So, we need to come up with a regular expression that characterizes these lines.  
       That is what we did using the option “pattern”.
    6. As we can see, most of the Stack Trace is indented by a tab.  
       We need to add the lines starting with “Caused by” to the regular expression as well as they too belong to the same previous event.
    7. We a