1. 
2. **Terms**:

Execution Models.

Inputs = Input Plugins.

Worker Queue.

Pipeline Workers = Batches = Filters + Output Plugins (Including Codec inside the Output Plugin).

Its own thread.

1. Now we know the basics of the **Logstash**.
2. Let’s take a moment to see how it works under **the hood**.
3. In particular, let’s see the **Execution Model** and how **Logstash** handles the incoming requests.

1. **Let’s understand**:

A picture containing text, screenshot, font, design

Description automatically generated

1. So, we have a number of inputs listening for events in one way or other.
2. Each **Input plugin** runs in its own **thread** to avoid **input plugins blocking** each other.
3. It means if we have two incoming events at the same time, they will be handled concurrently instead of one input having to wait until the other one finishes its work.
4. As we saw earlier, input plugin may use a codec to decode the incoming events such as decoding JSON, for instance.
5. After receiving **an event** and potentially applying the **codec**, the **input plugin** sends the event to the **Worker Queue**.
6. So, now all the received events are in **Worker Queue**.
7. From there (Worker Queue), **Logstash** uses the **Pipeline Workers** or **batches** to perform the rest of the work **involving filters and outputs plugins** along with the any **Codec** used within the **Output Plugin.**
8. Each **pipeline worker** or **batch** runs in its own thread meaning that multiple events can be processed simultaneously.
9. **A Pipeline Worker** consumers events from the **event queue** **in batches.**This is the way of optimizing the processing of events to increase the throughput of the pipeline as a whole.
10. The **batch size** is determined by the two **Configuration Options**
    1. **A number** representing the **maximum batch size** and
    2. **Batch Delay** : How long to wait before processing undersized batch of events.  
       Suppose, the maximum batch is set to 50 and   
       The batch delay = 100 milliseconds.  
       In this case, a batch of events will be processed if there 50 unprocessed events in the **work queue** or   
       if 100 milliseconds have elapsed.  
       The reason that a batch will be processed, even if the maximum batch size is not reached, is not to delay the processing by too much.  
       We still want events to be processed in a timely manner.  
       This is especially the case for pipelines that process the low volume of events, in which case we don’t want to keep the events in the **Work Queue** for too long.  
       **An Example** could be that we have a pipeline that processes **error logs** from web servers and pushes **those errors** to some **Dashboard**.  
       An example could be that we are **outputting the errors to Elastic Search** and uses **Kibana to display and analyse errors.  
       (There will be then a pipeline worker consisting of Elasticsearch Output Plugin responsible to send the processed events to Elasticsearch)**Hopefully, whatever web app is running on a web server doesn’t contain too many errors, so we should be dealing with a fairly low number of events.  
       In this case, it might take a long time to each 50 events but we want errors to be processed before reaching this threshold because otherwise there will be long delay before we see the errors appear within Kibana.  
       That is why small batches are processed when reaching the maximum delay.  
       This discussion was about the motivation for using maximum batch delay but why Logstash works with batches in the first Place.
11. **Why Logstash works with batches in the first Place?**
    1. One of the reasons is simply that some code needs to be executed regardless of how many events are processed at a time within a pipeline worker = Batch.

So, instead of executing that code 100 times for 100 events separately, it is more efficient to execute once for a batch of 100 events.

* 1. Another reason is that some output plugins are able to group together events as batches.   
     An example of this is the **Elastic Search Output Plugin.**Instead of sending 100 requests to **Elastic Search** being one for each event, the plugin uses the **Bulk API** that is exposed by the **Elastic Search.**So, to group together multiple events, so instead of sending 100 HTTP requests, a single request including all of the events is sent.  
       
     Please we are assuming that Batch Delay was not reached because otherwise individual requests will be sent.  
     For example: If the Batch Delay = 100 milliseconds and we send an event to Logstash per second, the output plugin will still be sending one request to Elastic Search per Second.  
     The default **batch size and batch delay** must work for most people as those will enable a very high number of events to be processed.  
     So we don’t need to change this default configuration until we find ourselves in a situation where we need to squeeze some more performance out of **Logstash**.

1. **Brief**:
   1. Logstash has a number of **Pipeline Workers running in a separate thread**.
2. But how many workers are started up?
3. It can be determined by startup by inspecting the hardware of the machine.  
   In particular, Logstash inspects the number of the CPU Cores and starts up an appropriate number of pipeline workers based on that.
4. There are a couple of reasons for this.
   1. It is useless to start up 5 workers on a machine having 2 CPU Cores because these workers can’t perform concurrently.
   2. On the other hand, only 5 workers on a machine having 10 CPU Cores would effectively limit the throughput of the Logstash instance because we will not be able to fully utilize the **Hardware Resources**.
5. With this default configuration by Logstash based on the processing power of the machine, we get the maximum and we don’t need to do anything.   
   See, the Local machine and the production machine don’t have the same processing power so inspecting the CPU Cores by the Logstash to start up that many pipeline workers make the right configuration to get maximum utilization.
6. After all this being said, it is possible to configure a fixed number of pipeline workers instead if we prefer.
7. One last thing:
   1. Logstash uses queues within each pipeline as well.
   2. More specifically between the stages such as between running filters and outputs.
   3. This is to buffer events that have completed one stage and waiting for the next one.
   4. This is all completely transparent to us as a user and so not important to know how internally it is implemented.
   5. So, Logstash uses in-memory queues by default so events will be lost if Logstash crashes due to any reasons.
   6. The chance of this happening is pretty slim unless we are dealing with a high volume of events, but it is worth keeping in mind.
   7. It is possible to change the queue to use the disk instead of memory.  
      In that case, events are persisted even then Logstash shuts down unexpectedly, yet the events have to still be processed through the pipeline.