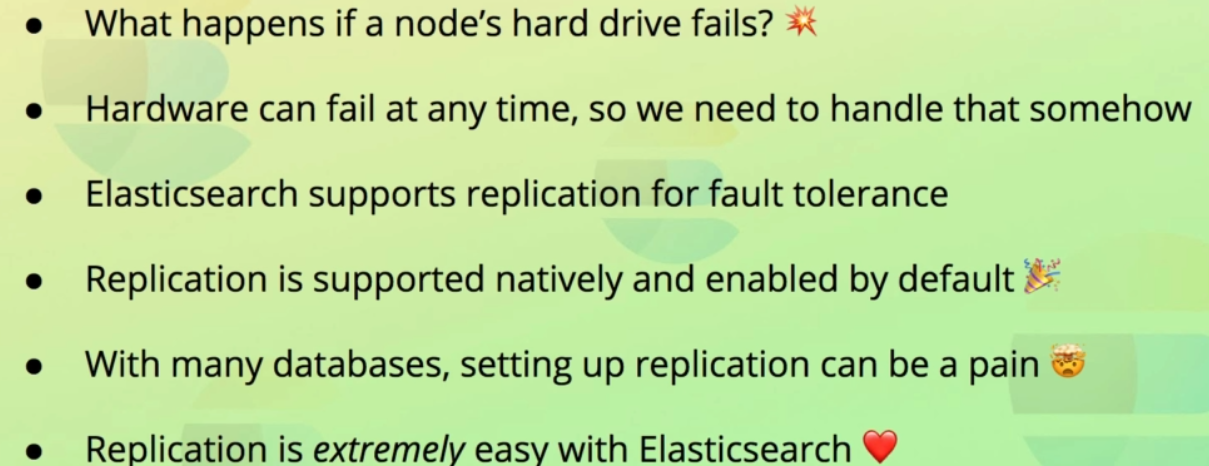
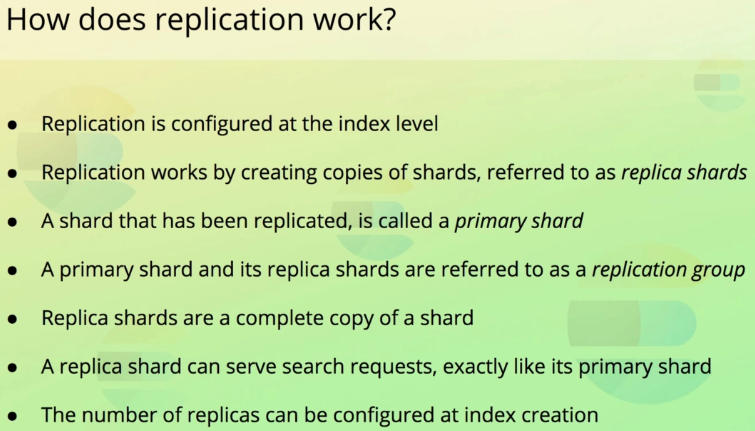
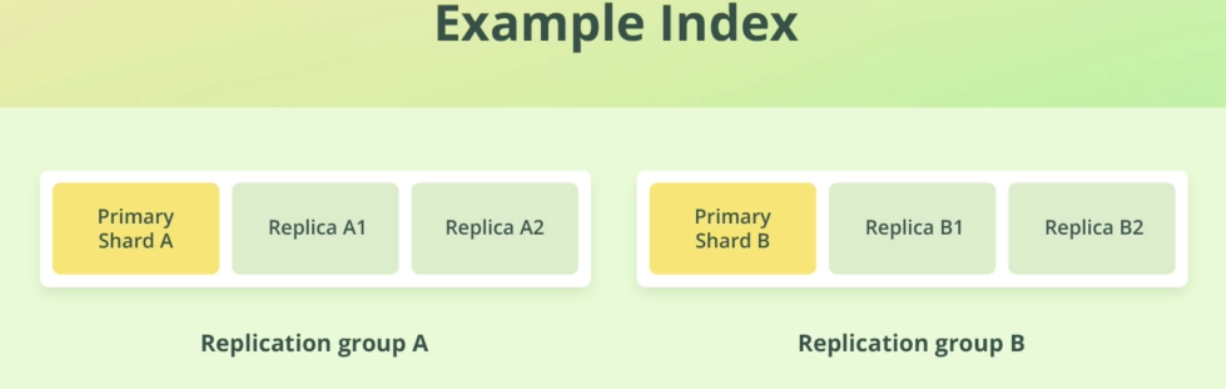
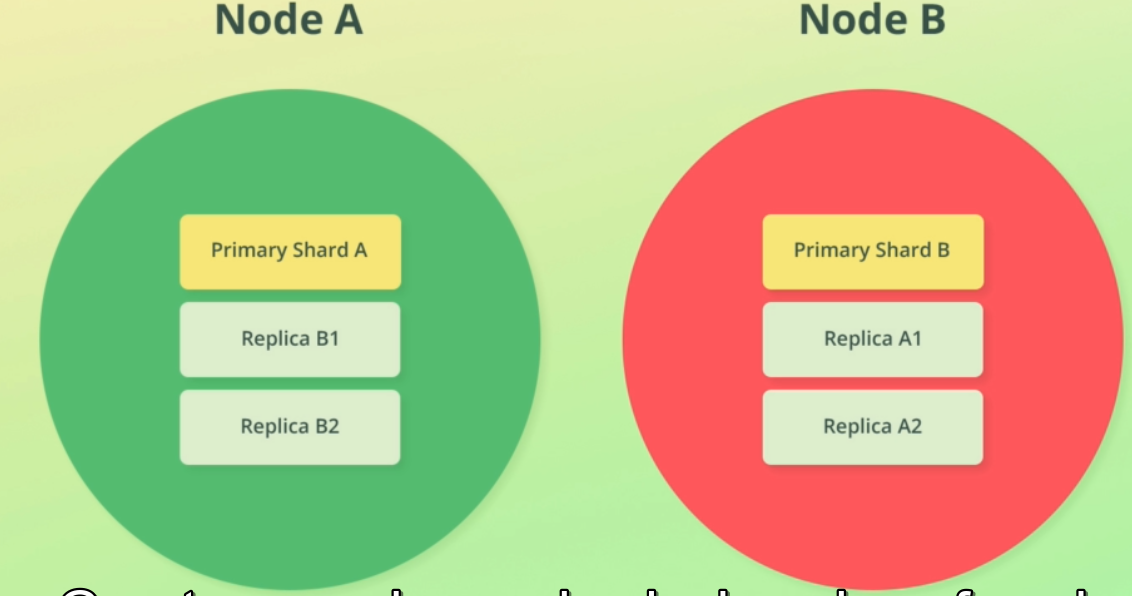
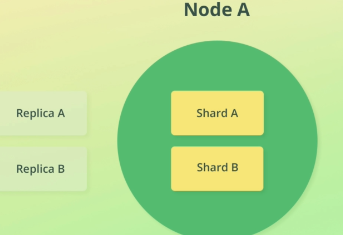
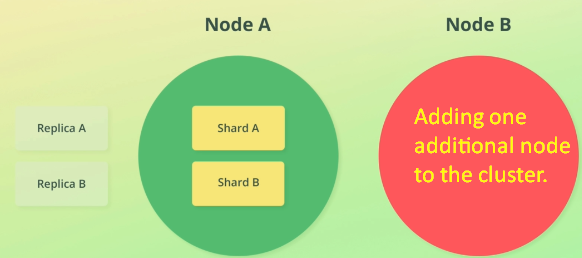
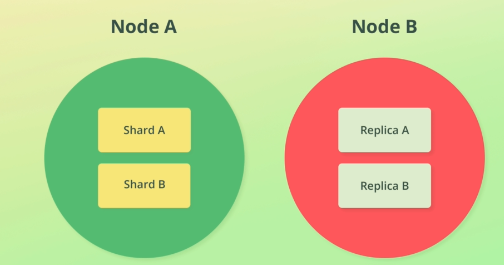
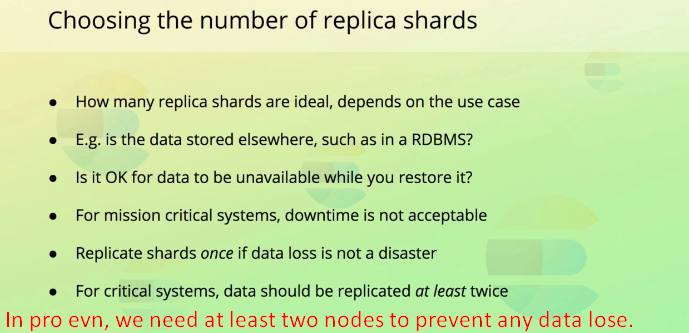
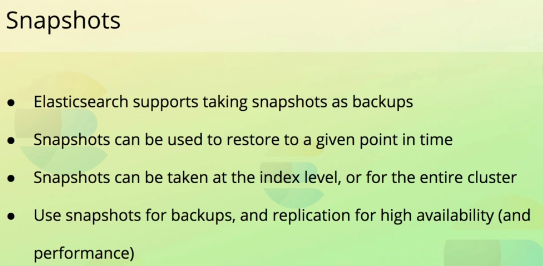
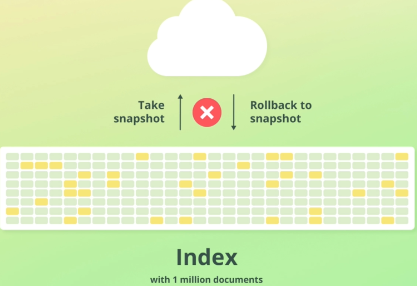
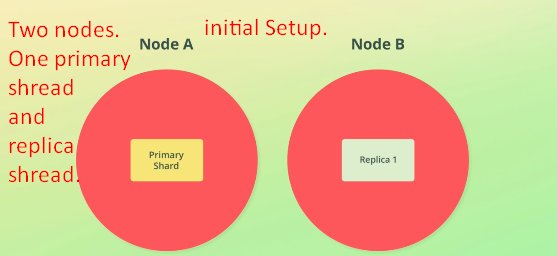
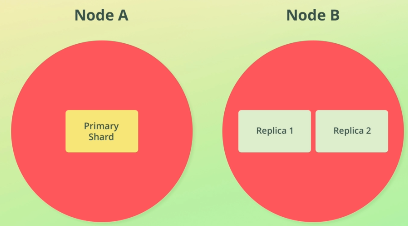
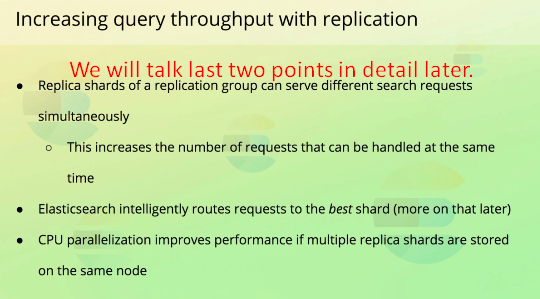
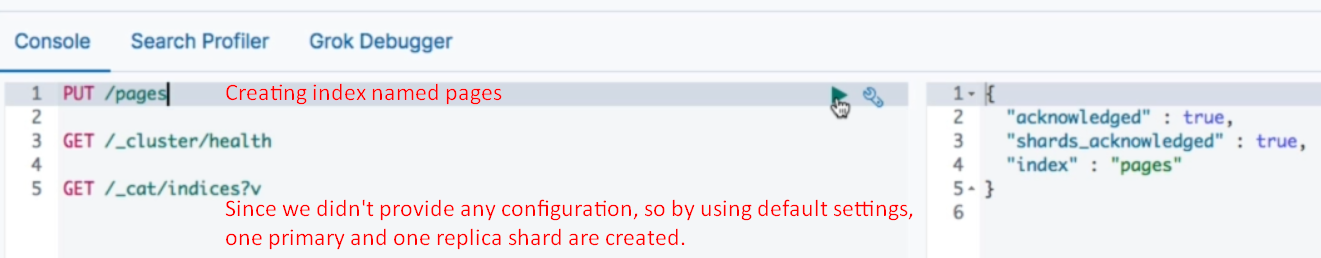
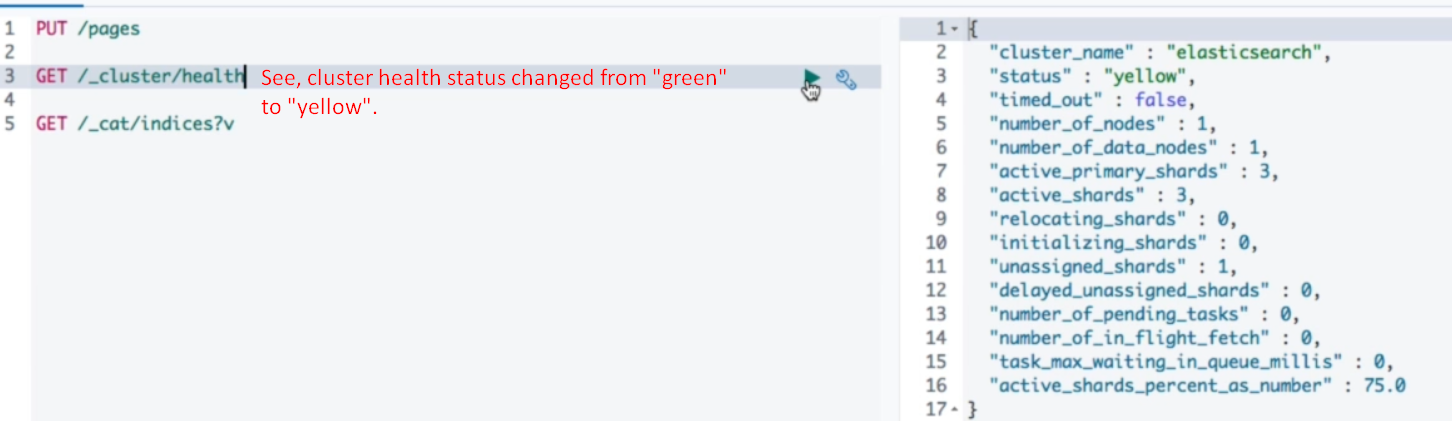
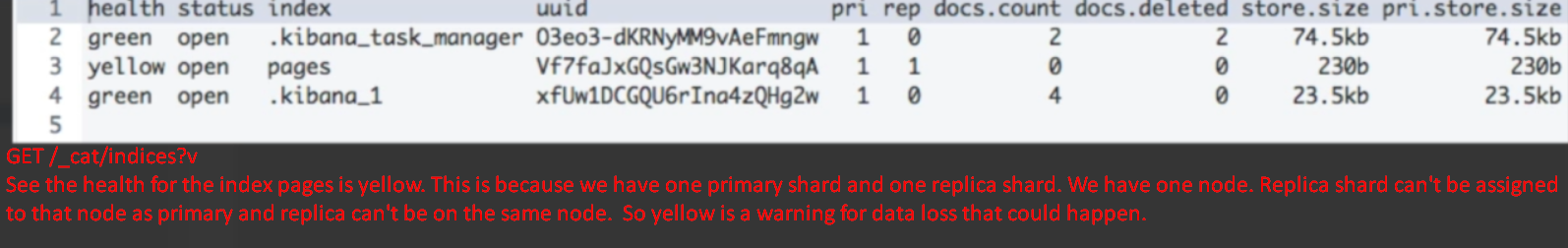
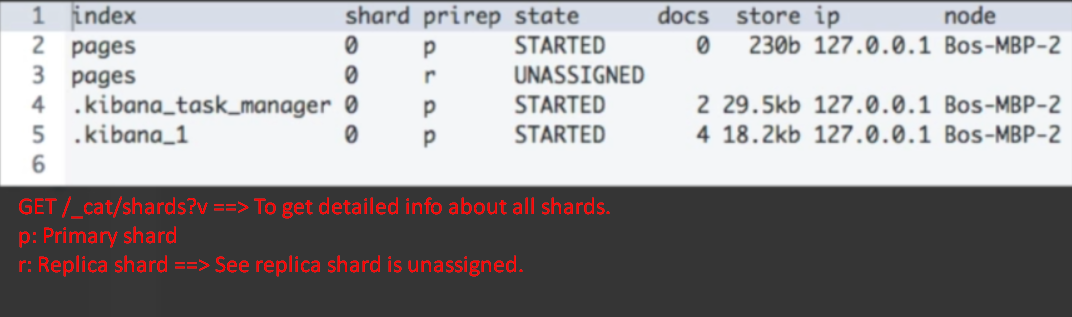
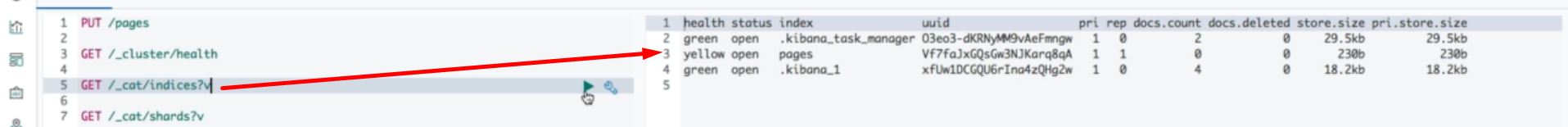
1. 
2. Let’s discuss replication closely related to sharding.
3. We know that by default index consists of one shard.   
   But what happens if the node where shard is stored is breaks down, i.e. has a disk failure?  
   

**Answer**: Data is lost since we don’t have copy of it.  
That is a major problem as hardware can fail at any given time.  
The more hard drives that are used to run your cluster, the higher the changes of a failure.  
Obviously this needs to be dealt with, because we want to be able to sleep at night, knowing that a hard drive failure is not a disaster.  
Therefore we need some fault tolerance and failover mechanism which is where **replication** comes into picture.

Elasticsearch natively supports replication of shards, and this is actually enabled by default, with zero configurations.

1. **So, how does replication work in Elasticsearch? **
   1. As we know, an index is configured to store its data within a number of shards, which may be stored across multiple nodes.  
      Likewise, replication is also configured at the index level.  
      Replication works by creating copies of each of the shards that an index contains.  
      These copies are referred to as **replicas or replica shards**.  
      **Primary Shard**: A shard that has been replicated.  
      **Replication Group**: Primary Shard + its replica shards.  
      Index contains shards which in turn contains replica shards.  
        
      In the above diagram, you can see one index contains two primary **shards and each primary shard contains two replica shards.**The **index** contains therefore **two replication group.**.  
        
      Great, so we have physical copies of each of our shards, but how does that help if the entire disk stops working, and we lose all of its data?  
      To prevent this from happing, replica shards are never stored on the same node as their primary shard.  
      It means if node disappears, there will always be at least one copy of a shard’s data available on different node.  
      How many copies will be left, depends on how many replicas are configured for the index and how many nodes the cluster contains.

Replication makes sense if there is more than one node otherwise replication is not going to help if the only available node breaks down.  
For this reason, Elasticsearch will only add replica shards for clusters with multiple nodes.  
You can still configure an index to contain one or more replicas for each shard, but it will not have any effect until an additional node is added.

1. **Let’s go through an example of how replication works within an Elasticsearch cluster**?  
   
   1. Let’s say we have two indices within the cluster.
   2. Both uses default configuration which one shard per index.
   3. Cluster has single node.
   4. Each index has one shard so the node will contain a total of two shards.
   5. Even though the indices are configured to replicate each shard once, the replica shards will be unassigned because we only have a single node running.   
      Jatin 🡺 As we know that replica shards are not put on the node having primary shard for those replica shards.
   6. This is fine for a development environment because it’s only inconvenient if we lose data but in product we can’t risk losing any data.
   7. So, we add one additional node. These nodes need not to be powerful at all. They just need to run on independent hardware so that there is no single point of failure.  
      
   8. Once Elasticsearch recognizes that we have added an additional node, it will enable replication meaning that the replica shards will be assigned.  
      
   9. Had we configured the indices to replicate shards twice, two replica shards would be placed on the other node instead, but the concept remains the same. Second node will have two replica shards for a primary shard because it is the rule that node containing primary shard doesn’t store replica.  
      If in future, we would add one more node to the cluster, we would see that the replica shards would be spread out to increase the **availability** even more. In that case we will not lose any data even if two nodes go down at the same time.
2. **Choosing the number of replica shards**.  
   
   1. Typically we would be fine with one or two replicas, but that depends on how critical your setup is.  
      If two nodes break down at the same time, can you then restore the data that was stored on them from another data source, such as RDBM.
   2. Is it acceptable for the data to be unavailable while you restore it?  
      If you’re using Elasticsearch for powering the search functionality on your personal wordpress blog, you will probably be fine with this very small risk.  
      On the other hand, if you’re using Elasticsearch for something critical within a hospital, you probably can’t afford to take that risk.
3. **Snapshots**:  
   
   1. Elasticsearch also supports taking snapshots just like many databases do.
   2. Snapshots provide a way to take backups so that you can restore data to a point in time.
   3. We can snapshot either a specific indices or the entire cluster.
   4. **Why do we need replication if we take snapshots**?
   5. Replication is indeed a way of **preventing data loss** but replication only works with “**live data**”. This essentially means that replication ensures that you will not lose the data that a given index stores at the current point in time.  
      Snapshots, on the other hand, enable you to export the current state of the cluster (or specific indices) to a file.  
      This file can then be used to restore the state of the cluster of indices to that state.
   6. **For instance**: 🡸 **Why we take snapshots even when we have replication**.  
        
      Suppose we’re tasked to restructure index having millions of documents and we are very much confident as we tested it on test server. But before running the queries, we take snapshots of the index so that we can recover from any implications.  
      When running the queries, things didn’t go as planned, perhaps because our test documents differed from the documents stored within the live index.  
      Whatever the cause, the documents got messed up, and we need to revert the changes to get the things back to the working state.  
      Replication can’t help with that because replication just ensures that we don’t lose our latest data which has already been modified in this example.  
      Instead, we need to revert the state of the index to the snapshot that we took.  
      Doing that, we should be all good and ready to try again after having fixed whatever went wrong.  
      This was the difference b/w snapshot and replication.  
      Snapshots are commonly used for daily backups and manual snapshots may be taken before applying changes to data, just to make sure that there is a way to roll back the changes in case something goes wrong.  
      Replication just ensures that indices can recover from a node failure and keep serving requests, as if nothing had happened.  
      Replication also help to increase the throughput of an index.  
      Suppose we have a web shop where we have the products stored within an index named “products”. We display the most popular products on the front page, and we also run the queries against the index when users search for products. This index gets queried a lot. Currently we have just one shard and one replica. But soon we start to experience a **performance bottleneck** for the queries run against the index at peak hours, so we need to find a way to handle that.  
        
      **Solution**: The initial thought could be to add additional node to the cluster which currently consists of two nodes. Having only primary shard and one replica shard would not help because we can’t spread these out across more than two nodes anyway.  
      First for us to utilize the additional node, we would have to increase the number of **replica shards**.  
      Adding one more replica would increase the throughput but not the availability.  
        
      As replica shard is fully functional index. Both of the replica shard can be queried at the same time.  
      This is possible because of two things;
      1. The fact that Elasticsearch automatically coordinates where queries will be executed
      2. Second parallelization. CPU has at least.   
         This means that the node hosting two replica shards can run a search query on each of the shards in parallel thus increasing the throughput of the index.  
         of course adding more replica will help in increasing throughput but not availability of data that too if the hardware resources of the node have not yet been fully utilized.  
         
4. Replica helps in achieving
   1. Throughput.
   2. Availability.
5. Let’s create a new index 🡺 **pages**.
6. 
7. 
8. 
9. 
10. 

See Kibana indices are configured with one shard and zero replica. The one shard makes sense because these indices will store very small amount of data.  
Don’t be fooled by zero though because adding another node to the cluster would increase to one.  
How?  
  
Because the Kibana indices are configured with a setting named auto\_expand\_replicas=”0-1”   
This setting dynamically changes the number of replicas depending on how many nodes our cluster contains.