**Scenario Based Questions**

* Your team uses the ELK stack for distributed logging. Recently, the Elasticsearch service has started running out of memory frequently. How would you approach diagnosing and resolving this issue?
* Deliverables: A step-by-step troubleshooting plan and a list of potential optimizations.

**Answer :**

**Step-by-Step Troubleshooting Plan**:

1. **Monitor Elasticsearch Resource Usage**:
   * Use **elasticsearch/\_cat/nodes?v** and **elasticsearch/\_cluster/stats** API to analyze node memory usage and cluster statistics.

A screenshot of a computer

Description automatically generated

A computer screen shot of a computer screen

Description automatically generated

* + Check Elasticsearch logs for memory-related warnings or errors.

1. **Analyze Query and Index Load**:
   * Use the **elasticsearch/\_tasks** API (**elasticsearch/\_tasks?detailed=true**) to identify long-running or expensive queries.

A screenshot of a computer program

Description automatically generated

* + Review the number of shards per index using the **elasticsearch/\_cat/indices?v** API. Too many shards consume excessive memory.

A screenshot of a computer

Description automatically generated

1. **Examine Indexing and Data Retention**:
   * Check the size of each index and retention policies. Large indices might cause memory pressure.
   * Use the **elasticsearch/\_cat/allocation?v** API to assess disk usage and shard allocation.

A screenshot of a computer

Description automatically generated

1. **Adjust JVM Heap Size**:
   * Verify the Elasticsearch JVM heap settings (-Xms and -Xmx). Ideally, this should be 50% of the total memory, but not more than 32GB.
   * My configuration is on docker-compose.yml

A computer screen shot of a program

Description automatically generated

1. **Investigate Field Mapping**:
   * Too many fields or incorrect mappings (like text for high cardinality fields) can lead to memory issues.
   * Use the **elasticsearch/\_mapping** API to check field types and mappings.

A computer screen shot of a computer program

Description automatically generated

1. **Audit Node Capacity**:
   * Verify that node hardware (CPU, RAM, and disk I/O) matches the cluster workload.

**List of Potential Optimizations**:

1. **Shard and Index Management**:
   * Merge small shards using the **\_shrink** API.
   * Use rollover indices for time-series data to limit the number of active shards.
2. **Optimize Mappings and Indexing**:
   * Disable **\_source** and **\_all fields** if not needed.
   * Use the keyword type for exact-match fields instead of text.
3. **Implement Data Retention Policies**:
   * Delete or archive old data using the Index Lifecycle Management (ILM) feature.
4. **Cache Management**:
   * Tune cache settings like indices.fielddata.cache.size to avoid excessive memory usage.
   * Use query caching to reduce repetitive processing.
5. **Scale the Cluster**:
   * Add nodes to the cluster to distribute memory load.
6. **Monitor and Alerts**:
   * Use alerts to detect high memory usage early.