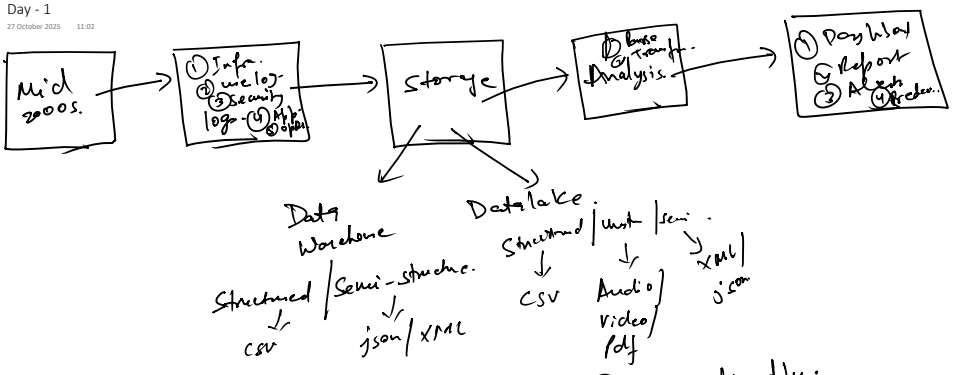
Day - 1

27 October 2025 11:02



1. Basic Appl / Archi → tool.
2. Market Demand / Popularity
3. Community Support.
4. License Cost.
5. Integration with App.
6. User friendly.
7. Scalability.
8. Customer Support
9. Analysis Engine.

Elastic Search :-

index → logical division.

Data → json. → (Key - value)

{ name : vivek Arora. }

Key value.

Components of Index :-

1. Document. → json record.
2. Fields. → Key - value.
3. Mapping. → Analyzed & stored.
4. Shard. → logical division of index for parallel Processing.
5. Replica. → copy of data.

Primary → original

Replica → copy

Common Indices operations :-

1. List all indices. → GET _cat/indices?v
2. Create an indices. → PUT / my-index
3. Delete an indices. → DELETE / my-index.
5. view index setting. → GET / my-index / _setting.
6. Add document → POST / my-index / _doc / 1 → id=1
7. Search Document → GET / my-index / _search ? q = user : john
8. view index Mapping → GET / my-index / _mapping.

Create index :-

PUT | Product

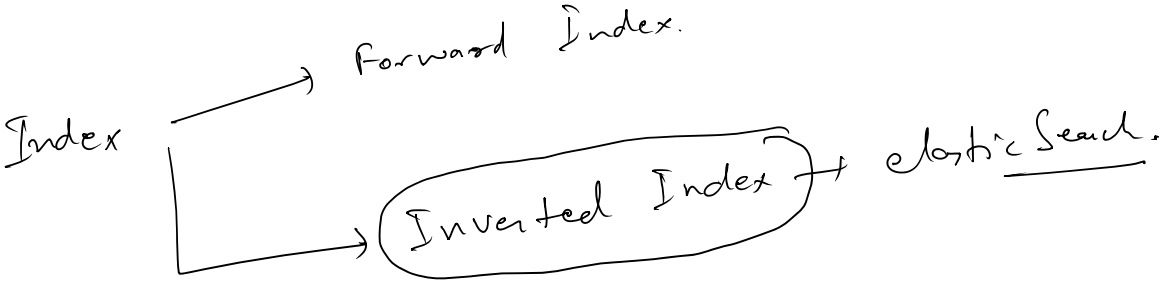
Add document :-

POST | Product | _doc | 1

```
{
  "name" : "dell" ,
  "price" : 55000,
  "place" : "BLR"
}
```

Search Docum :- GET | Product | _search

```
{ "query": {
    "name": "dell"
  }
}
```



Document ID	Content
1	"Elastic Search is fast"
2	"Elastic Search scales Horizontally"

Forward Index :- ID → Term.

1 → Elastic Search is fast

2 → Elastic Search Scales horizontally

Inverted Index :- Term → ID

, [1, 2]

Inverted Index:-

Term	→
ElasticSearch	→ [1, 2]
is	→ [1]
fast	→ [1]
Scales	→ [2]
Horizontally	→ [2]

How elasticSearch Builds:-

(1) Tokenization. – Text is split into token (Word) → ["ElasticSearch", "Scales", "Horizontally"]
(2) Normalization – Token into lowercase, stop word Removed, stemming is applied.
(3) Inverted Index Creation – Map term with ID.

"ElasticSearch is fast"

(1) Tokenization → ["ElasticSearch", "is", "fast"]
(2) lowercasing → ["elasticsearch", "is", "fast"]
(3) stop-word Removal → [is, a, an, the, are, were, was]

["elasticsearch", "fast"]

(4) Stemming:- Reduce the word to the root form.

ex → fastest → fast
running → run
runner → run

["elasticsearch", "fast"]

Benefit:-

(1) Reduce the index size → reduce noise.
(2) Improve Recalls → Running → Ran → run.
Precision → Ignore irrelevant filler word.

(2.) Improve [illegible]

(2) Improve Precision → Ignore irrelevant [illegible]

Filler Word [a, an, the, are, was, were, is, am, from, at, as]

① TF – Term frequency,
② IDF – Inverse Document frequency.

① Term frequency (TF) :– How often the term occur in the document.

$$TF(t,d) = \left(\frac{\text{Number of times term } t \text{ occurs in doc } d}{\text{Total no. of the terms in } d}\right)$$

② IDF (Inverted document frequency) :–

$$IDF = \left(\frac{\text{Total no. of documents}}{\text{No. of documents that contain the term } t}\right)$$

Combined Score = TF × IDF

D1 = "ElasticSearch is fast"
D2 = "Elastic search scales horizontally"
D3 = "Fast System Scales"

	TF(D1)	TF(D2)	TF(D3)	IDF
elasticSearch =	1	1	0	— 3/2 = 1.5
fast =	1	0	1	— 3/2 = 1.5

What is benefit?

(1)

TF - IDF

"Signal to the Noise ratio"

TF-IDF → Combined Score

↓

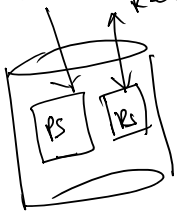
Story & clear picture.

* Elastic Search Scaling :-

Components :-

(1) cluster. — Collection of Nodes is called cluster.
(2) Node. — Sigle instance where elastic Search is running.
(3) Index — Database
(4) Shards — A physical piece of index.
(5) Replica — Copy the data from primary index.

ex- Index = VK-movies.
PrimaryShards = 2
replica = 1

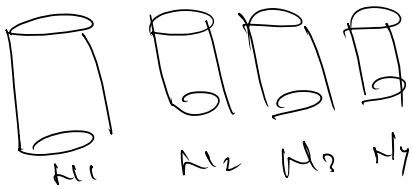


N1

Parallel write & Read concept.

How Query Scale?

1. Coordinating node that will receive Query.
2. Query sent to all the relevant shards.
3. Execute Query – fetch result.
4. Combine o/p → give result.



Cluster Benefits:

1. Availability
2. Handle huge data.
3. Performance
4. Query speed.

cluster.
- Node 1 – P1, R1
- Node 2 – P2, R2
- Node 3 – P3, R3

Index – logical division of data.

Shards – Physical division of data into smaller unit.

↳ fully functional Lucene index.

↓

Search library that powers the search engine like elastic search, solr

etc.

Why!

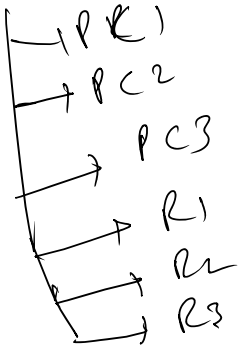
④ Performance..

① Scalability
② Availability
③ High Availability

Ex:- PUT | Customer_data.

```
{
  "setting": {
      "No. of shards": 3,
      "No of replica": 1
  }
}
```

Index: Customer_data.



* ILM (Index lifecycle management):-

Automate the index creation – deletion on the basis of index age, size & No. of documents.

① Recovery.
② Define which data from which index to retain
③ Adhere company policy.
④ free-up disk space.

— ① Hot – Read/write
Read ... data

4 Phases :-
1. Hot – Read
2. Warm – Read
3. cold – Rarely Query data
4. Delete – Deletion of data will happen.

Practical :-
1. Policy.
2. Template.
3. Index
4. Attach the template with index.

Analyzer :- Process the text during indexing & searching time.

Components :-
1. character filters.
2. Tokenizers.
3. Token filters.

Types of Built in Analyzers :-
1. standard.
2. Simple.
3. whitespace
4. language
5. stop
6. keyword
7. Pattern.

```
POST my_index/_analyze
{
  "analyzer" : "English"
  "text" : "elasticSearch scales horizontally"
}
```