Elasticsearch — Cheat Sheet

9.x line (8.19 = last 8.x) · verified against elastic.co/docs · irurueta.com

REQUEST SHAPE (DEV TOOLS / CURL)

```
# Kibana Dev Tools console
GET /my-index/_search
{ "query": { "match_all": {} } }
# curl equivalent
curl -sk -u elastic:pw \
  -H 'Content-Type: application/json' \
  https://localhost:9200/my-index/_search \
  -d '{"query":{"match_all":{}}}'

?pretty          human-readable JSON
?filter_path=hits.hits._source   trim body
?human   ?error_trace   ?format=yaml
```

CREATE INDEX: SETTINGS + MAPPINGS

```
PUT /books
{
  "settings": {
    "number_of_shards": 1,
    "number_of_replicas": 1,
    "refresh_interval": "1s"
  },
  "mappings": {
    "properties": {
      "title":     { "type": "text" },
      "isbn":      { "type": "keyword" },
      "pages":     { "type": "integer" },
      "published": { "type": "date" }
    }
  }
}
GET /books/_mapping
PUT /books/_mapping   add new fields only
```

TEXT VS KEYWORD + MULTI-FIELD

```
text     analyzed → full-text; not sort/agg
keyword  exact term; sort, aggregate, filter

"name": {
  "type": "text",
  "fields": {
    "raw": { "type": "keyword", "ignore_above":
256 }
  }
}
# match on: name   ·   sort/agg on: name.raw
```

ANALYSIS CHAIN

```
char filters → tokenizer → token filters → terms

"my QUICK brown FOXES!"
  standard tokenizer   [my, QUICK, brown, FOXES]
  lowercase            [my, quick, brown, foxes]
  stop                 [quick, brown, foxes]
  stemmer (english)    [quick, brown, fox]

POST /_analyze
{ "analyzer": "english",
  "text": "my QUICK brown FOXES!" }
# custom: settings.analysis.
{analyzer,filter,...}
```

DOCUMENT CRUD + _BULK

```
POST /books/_doc                auto-id, create
PUT  /books/_doc/1              index (replace)
PUT  /books/_create/1           fail if exists
GET  /books/_doc/1
POST /books/_update/1   { "doc": { "pages": 480
} }
POST /books/_update/1
  { "script": { "source": "ctx._source.pages++"
} }
DELETE /books/_doc/1

POST /_bulk   NDJSON, one action per line,
trailing \n
{ "index":  { "_index": "books", "_id": "1" } }
{ "title": "ES", "pages": 400 }
{ "update": { "_index": "books", "_id": "1" } }
{ "doc": { "pages": 420 } }
{ "delete": { "_index": "books", "_id": "2" } }
# check "errors" + per-item status in response
```

OPTIMISTIC CONCURRENCY CONTROL

```
GET /books/_doc/1   →  _seq_no, _primary_term

PUT /books/_doc/1?if_seq_no=4&if_primary_term=1
{ "title": "..." }

# 409 version_conflict_engine_exception on stale
# retry: re-GET → re-apply → re-PUT
# _update / _bulk update: ?retry_on_conflict=3
# external versioning: ?
version=7&version_type=external
```

_SEARCH: QUERY VS FILTER CONTEXT

```
GET /books/_search
{
  "query": {
    "bool": {
      "must":   [ { "match": { "title": "search"
} } ],
      "filter": [
        { "term":  { "isbn": "9781XXXX" } },
        { "range": { "pages": { "gte": 200 } } }
      ]
    }
  },
  "from": 0, "size": 10,
  "_source": [ "title", "pages" ]
}
# must/should → scored ; filter/must_not → no
_score, cached
```

QUERY SKELETONS

```
match        { "match": { "title": "quick brown"
} }
match_phrase { "match_phrase": { "title": "quick
brown" } }
multi_match  { "multi_match": { "query": "es",
   "fields": [ "title^3", "body" ],
   "type": "best_fields" } }
term         { "term":  { "status": "published"
} }
terms        { "terms": { "tag": [ "a", "b" ] }
}
range        { "range": { "pages":
                 { "gte": 100, "lt": 500 } } }
exists       { "exists": { "field": "isbn" } }
bool         must / should / filter / must_not
             "minimum_should_match": 1
```

FUNCTION_SCORE + SORT + SEARCH_AFTER / PIT

```
"function_score": {
  "query": { "match": { "title": "es" } },
  "functions": [
    { "filter": { "term": { "promoted": true }
}, "weight": 2 },
    { "field_value_factor": { "field": "rating",
        "modifier": "log1p" } } ],
  "boost_mode": "sum", "score_mode": "multiply"
}
"sort": [ { "published": "desc" }, "_score" ]

# deep paging: search_after + point-in-time
POST /books/_pit?keep_alive=1m          → id
GET /_search
{ "pit": { "id": "...", "keep_alive": "1m" },
  "sort": [ {"published":"desc"},
{"_shard_doc":"asc"} ],
  "search_after": [ 1700000000000, 42 ] }
```

AGGREGATIONS

```
GET /sales/_search
{
  "size": 0,
  "aggs": {
    "by_month": {
      "date_histogram": {
        "field": "date",
        "calendar_interval": "month"
      },
      "aggs": {
        "revenue": { "sum": { "field": "amount"
} }
      }
    },
    "top_tags": {
      "terms": { "field": "tag", "size": 10 }
    }
  }
}
# metrics: sum avg min max stats cardinality
percentiles
# bucket: terms histogram date_histogram range
filters nested
```

NESTED VS JOIN

```
nested — array of objects, indexed on one shard
"comments": { "type": "nested" }
{ "nested": { "path": "comments",
    "query": { "match":
      { "comments.text": "great" } } } }

join — separate parent + child docs, same shard
"my_join": { "type": "join",
  "relations": { "book": "review" } }
# index child with ?routing=1
{ "my_join": { "name": "review", "parent": "1" }
}
query: has_child / has_parent / parent_id
```

VECTOR: KNN + RETRIEVER (HYBRID)

```
"embedding": { "type": "dense_vector", "dims":
384,
  "index": true, "similarity": "cosine" }

GET /docs/_search
{ "knn": { "field": "embedding",
    "query_vector": [ 0.1, 0.2, "..." ],
    "k": 10, "num_candidates": 100 },
  "size": 10 }

# hybrid: retriever + RRF
{ "retriever": { "rrf": { "retrievers": [
    { "standard": { "query":
        { "match": { "body": "es" } } } },
    { "knn": { "field": "embedding",
        "query_vector": [ "..." ],
        "k": 50, "num_candidates": 200 } }
  ], "rank_window_size": 50 } } }
```

ES|QL ONE-LINER

```
POST /_query
{ "query": "FROM sales
    | WHERE region == \"EU\"
    | STATS revenue = SUM(amount) BY tier
    | SORT revenue DESC
    | LIMIT 10" }
# piped, tabular result; also Kibana Discover /
ES|QL
```

_CAT ONE-LINERS

```
GET _cat/health?v
GET _cat/indices?v&s=store.size:desc
GET _cat/shards?v
GET _cat/nodes?v        GET _cat/aliases?v
GET _cat/allocation?v   GET _cat/thread_pool?v
# ?v headers · ?s= sort · ?format=json · ?
h=col,col
GET _cluster/health
GET _cluster/allocation/explain
```

ALIAS SWAP + ILM PHASES

```
# atomic, zero-downtime cutover
POST /_aliases
{ "actions": [
  { "remove": { "index": "books-v1", "alias":
"books" } },
  { "add":    { "index": "books-v2", "alias":
"books" } }
] }
# rolling write target: "is_write_index": true

ILM:  hot → warm → cold → frozen → delete
hot     rollover (max_age /
max_primary_shard_size / max_docs)
warm    forcemerge, shrink, allocate
cold    searchable_snapshot (full copy)
frozen  searchable_snapshot (shared cache)
delete  delete    # min_age gates each phase
```

SNAPSHOT + RESTORE

```
PUT /_snapshot/my_repo
{ "type": "fs",
  "settings": { "location": "/mnt/backups" } }

PUT /_snapshot/my_repo/snap-1?
wait_for_completion=true
{ "indices": "books,sales",
  "include_global_state": false }

POST /_snapshot/my_repo/snap-1/_restore
{ "indices": "books",
  "rename_pattern": "(.+)",
  "rename_replacement": "restored-$1" }

# incremental (new segments only) · GET .../_all
# scheduled: PUT /_slm/policy/daily-snapshots
```

DECIDE: QUERY VS FILTER CONTEXT

```
Filter context   yes/no match · cached · no
_score
  → term, terms, range on exact fields; dates;
    enums; status flags; ACL / tenant tags
Query context    "how well" · contributes to
_score
  → match, multi_match, match_phrase on analyzed
text
Rule   put every non-scoring clause under
       bool.filter (or bool.must_not)
```