Introduction

- We will cover the most important mapping parameters
- This is not an exhaustive list, though
    - Most other parameters are very specialized and/or rarely used

`format` parameter

- Used to customize the format for `date` fields
- I recommend using the default format whenever possible
  - `"strict_date_optional_time||epoch_millis"`
- Using Java's `DateFormatter` syntax
  - E.g. `"dd/MM/yyyy"`
- Using built-in formats
  - E.g. `"epoch_second"`

`format` parameter (examples)

```
PUT /sales
{
  "mappings": {
    "properties": {
      "purchased_at": {
        "type": "date",
        "format": "dd/MM/yyyy"
      }
    }
  }
}
```

```
PUT /sales
{
  "mappings": {
    "properties": {
      "purchased_at": {
        "type": "date",
        "format": "epoch_second"
      }
    }
  }
}
```

`properties` parameter

- Defines nested fields for `object` and `nested` fields

```
PUT /sales
{
  "mappings": {
    "properties": {
      "sold_by": {
        "properties": {
          "name": { "type": "text" }
        }
      }
    }
  }
}
```

```
PUT /sales
{
  "mappings": {
    "properties": {
      "products": {
        "type": "nested",
        "properties": {
          "name": { "type": "text" }
        }
      }
    }
  }
}
```

`coerce` parameter

- Used to enable or disable coercion of values (enabled by default)

```
PUT /sales
{
  "mappings": {
    "properties": {
      "amount": {
        "type": "float",
        "coerce": false
      }
    }
  }
}
```

```
PUT /sales
{
  "settings": {
    "index.mapping.coerce": false
  },
  "mappings": {
    "properties": {
      "amount": {
        "type": "float",
        "coerce": true
      }
    }
  }
}
```

Introduction to `doc_values`

- Elasticsearch makes use of several data structures
  - No single data structure serves all purposes
- Inverted indices are excellent for searching text
  - They don’t perform well for many other data access patterns
- “Doc values” is another data structure used by Apache Lucene
  - Optimized for a different data access pattern (document → terms)

Introduction to `doc_values`

- Essentially an “uninverted” inverted index
- Used for sorting, aggregations, and scripting
- An *additional* data structure, not a replacement
- Elasticsearch automatically queries the appropriate data structure

Disabling `doc_values`

- Set the `doc_values` parameter to `false` to save disk space
  - Also slightly increases the indexing throughput
- Only disable doc values if you won't use aggregations, sorting, or scripting
- Particularly useful for large indices; typically not worth it for small ones
- Cannot be changed without reindexing documents into new index
  - Use with caution, and try to anticipate how fields will be queried

doc_values parameter (example)

```
PUT /sales
{
  "mappings": {
    "properties": {
      "buyer_email": {
        "type": "keyword",
        "doc_values": false
      }
    }
  }
}
```

`norms` parameter

- Normalization factors used for relevance scoring
- Often we don't just want to *filter* results, but also *rank* them
- Norms can be disabled to save disk space
    - Useful for fields that won't be used for relevance scoring
    - The fields can still be used for filtering and aggregations

`norms` parameter (example)

```
PUT /products
{
  "mappings": {
    "properties": {
      "tags": {
        "type": "text",
        "norms": false
      }
    }
  }
}
```

`index` parameter

- Disables indexing for a field
- Values are still stored within `_source`
- Useful if you won't use a field for search queries
- Saves disk space and slightly improves indexing throughput
- Often used for time series data
- Fields with indexing disabled can still be used for aggregations

`index` parameter (example)

```
PUT /server-metrics
{
  "mappings": {
    "properties": {
      "server_id": {
        "type": "integer",
        "index": false
      }
    }
  }
}
```

`null_value` parameter

- `NULL` values cannot be indexed or searched
- Use this parameter to replace `NULL` values with another value
- Only works for explicit `NULL` values
- The replacement value must be of the same data type as the field
- Does not affect the value stored within `_source`

`null_value` parameter (example)

```
PUT /sales
{
  "mappings": {
    "properties": {
      "partner_id": {
        "type": "keyword",
        "null_value": "NULL"
      }
    }
  }
}
```

`copy_to` parameter

- Used to copy multiple field values into a “group field”
- Simply specify the name of the target field as the value
- E.g. `first_name` and `last_name` → `full_name`
- *Values* are copied, not terms/tokens
  - The analyzer of the target field is used for the values
- The target field is not part of `_source`

`copy_to` parameter (example)

```
PUT /sales
{
  "mappings": {
    "properties": {
      "first_name": {
        "type": "text",
        "copy_to": "full_name"
      },
      "last_name": {
        "type": "text",
        "copy_to": "full_name"
      },
      "full_name": {
        "type": "text"
      }
    }
  }
}
```

```
POST /sales/_doc
{
  "first_name": "John",
  "last_name": "Doe"
}
```

CodingExplained
CODE. SLEEP. REPEAT.